SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Gitcoin
Prepared by:	Sherlock
Lead Security Expert:	WATCHPUG
Dates Audited:	September 11 - September 21, 2023
Prepared on:	November 2, 2023

Introduction

A set of smart contracts that enable democratic allocation and distribution of capital. Developed by Gitcoin to power the Grants Stack, but useful beyond grants and quadratic funding.

Scope

Repository: allo-protocol/allo-v2

Branch: main

Commit: 0b881ef4a0013d2809374c9ea69f4cf1288dfe62

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
16	5

Security experts who found valid issues

jkoppel
0xnirlin
0xkaden
Kow
nobody2018
GimelSec
branch_indigo
WATCHPUG
sashik_eth
honeymewn
rvierdiiev
0xdeadbeef
lemonmon
Arz
alexxander
KingNFT
zach030
pinalikefruit
0xG0P1
niluk
VAD37
shirochan
qbs
0xbepresent
jah
detectiveking
HChang26

fibonacci
osmanozdemir1
ast3ros
HHK
BenRai
simon135
0x180db
0x00ffDa
hals
xAriextz
ArmedGoose
gkrastenov
ace13567
0xc0ffEE
pengun
tnquanghuy0512
carrotsmuggler
imsrybr0
Martians
Kral01
lil.eth
wangxx2026
vangrim
ZdravkoHr.
p0wd3r
SBSecurity
AsenXDeth
trachev
imare
Silvermist
alymurtazamemon
Shubham
RadCet
0xarno
bronze_pickaxe
twcctop
Proxy
B353N
pontifex
fishgang
dany.armstrong90
grearlake
ashirleyshe
toshii
Nyx
0xMAKEOUTHILL
Vagner
hindsight
unix515
ustas
seeques
jovi
chaduke
coffiasd
0x3b
Kodyvim
sandNallani
cu5t0mPe0
dipp
al88nsk
shtesesamoubiq
Topmark
0xRstStn
0xpoolbaer
0xMosh
DevABDee
vagrant
cats
JP_Courses
Aamirusmani1552
Inspex
inzinko
0xHelium
pavankv241
0x6980
marchev
lealCodes
inspecktor
adeolu
alexzoid
Tri-pathi
0xgoat
foresthalberd
parsely
trevorjudice
0x1337
theclonedtyroneidgafmf
tsvetanovv

Issue H-1: `QVSimpleStrategy` never updates `allocator.voiceCredits`.

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/150

Found by

0x00ffDa, 0x3b, 0xMAKEOUTHILL, 0xarno, 0xbepresent, 0xkaden, Arz, BenRai, GimelSec, HChang26, HHK, Kodyvim, Kow, Kral01, Martians, Nyx, WATCHPUG, ZdravkoHr., al88nsk, alexxander, ashirleyshe, ast3ros, bronze_pickaxe, carrotsmuggler, chaduke, coffiasd, cu5t0mPe0, dany.armstrong90, detectiveking, dipp, fibonacci, jah, jkoppel, jovi, lemonmon, lil.eth, nobody2018, osmanozdemir1, pengun, pontifex, qbs, rvierdiiev, sandNallani, seeques, simon135, tnquanghuy0512, toshii, wangxx2026 Every allocator in `QVSimpleStrategy` has a maximum credit limit. An allocator should not be able to bypass the limit. However, `QVSimpleStrategy` fails to record the allocated votes. An allocator can vote as many as possible.

Vulnerability Detail

`QVSimpleStrategy._allocate` calls `_hasVoiceCreditsLeft` to check that the recipient has voice credits left to allocate. https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-simple/QVSimpleStrategy.sol#L121

```
function _allocate(bytes memory _data, address _sender) internal virtual
↪   override {


    // check that the recipient has voice credits left to allocate
    if (!_hasVoiceCreditsLeft(voiceCreditsToAllocate, allocator.voiceCredits))
↪   revert INVALID();

    _qv_allocate(allocator, recipient, recipientId, voiceCreditsToAllocate,
↪   _sender);
}
```

`QVSimpleStrategy._hasVoiceCreditsLeft` checks `_voiceCreditsToAllocate + _allocatedVoiceCredits <= maxVoiceCreditsPerAllocator` https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-simple/QVSimpleStrategy.sol#L144

```
function _hasVoiceCreditsLeft(uint256 _voiceCreditsToAllocate, uint256
↪   _allocatedVoiceCredits)
    internal
    view
    override
```

```
    returns (bool)
{
    return _voiceCreditsToAllocate + _allocatedVoiceCredits <=
↪   maxVoiceCreditsPerAllocator;
}
```

The problem is that `allocator.voiceCredits` is always zero. Both `QVSimpleStrategy` and `QVBaseStrategy` don't update `allocator.voiceCredits`. Thus, allocators can cast more votes than `maxVoiceCreditsPerAllocator`.

Impact

Every allocator has an unlimited number of votes.

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-simple/QVSimpleStrategy.sol#L121
https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-simple/QVSimpleStrategy.sol#L144

Tool used

Manual Review

Recommendation

Updates `allocator.voiceCredits` in `QVSimpleStrategy._allocate`.

```
    function _allocate(bytes memory _data, address _sender) internal virtual
↪   override {
        ...

        // check that the recipient has voice credits left to allocate
        if (!_hasVoiceCreditsLeft(voiceCreditsToAllocate,
↪   allocator.voiceCredits)) revert INVALID();
+       allocator.voiceCredits += voiceCreditsToAllocate;
        _qv_allocate(allocator, recipient, recipientId, voiceCreditsToAllocate,
↪   _sender);
    }
```

Discussion

osmanozdemir1

Escalate

This issue should be High. Allocators can have unlimited credit and easily manipulate the whole voting.

sherlock-admin2

> Escalate
>
> This issue should be High. Allocators can have unlimited credit and easily manipulate the whole voting.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

neeksec

Agree with the Escalation.

MLON33

https://github.com/allo-protocol/allo-v2/pull/339 https://github.com/allo-protocol/allo-v2/commit/1efc544fdf988896c89dc2056f7efa114b83aab9

Evert0x

Planning to accept escalation and assign high severity

Evert0x

Result: High Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- osmanozdemir1: accepted

Issue H-2: `recipientsCounter` should start from 1 in `DonationVotingMerkleDistributionBaseStrategy`

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/199

Found by

0xkaden, 0xnirlin, GimelSec, Kow, branch_indigo, nobody2018 When doing `DonationVotingMerkleDistributionBaseStrategy._registerRecipient`, it checks the current status of the recipient. If the recipient is new to the pool, the status should be `Status.None`. However, `recipientsCounter` starts from 0. The new recipient actually gets the status of first recipient of the pool.

Vulnerability Detail

`DonationVotingMerkleDistributionBaseStrategy._registerRecipient` calls `_getUintRecipientStatus` to get the current status of the application. The status of the new application should be `Status.None`. Then, the `recipientToStatusIndexes[recipientId]` to `recipientsCounter` and `recipientsCounter`. https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/donation-voting-merkle-base/DonationVotingMerkleDistributionBaseStrategy.sol#L580

```
function _registerRecipient(bytes memory _data, address _sender)
    internal
    override
    onlyActiveRegistration
    returns (address recipientId)
{


    uint8 currentStatus = _getUintRecipientStatus(recipientId);

    if (currentStatus == uint8(Status.None)) {
        // recipient registering new application
        recipientToStatusIndexes[recipientId] = recipientsCounter;
        _setRecipientStatus(recipientId, uint8(Status.Pending));

        bytes memory extendedData = abi.encode(_data, recipientsCounter);
        emit Registered(recipientId, extendedData, _sender);

        recipientsCounter++;
    } else {
        if (currentStatus == uint8(Status.Accepted)) {
            // recipient updating accepted application
            _setRecipientStatus(recipientId, uint8(Status.Pending));
```

```
        } else if (currentStatus == uint8(Status.Rejected)) {
            // recipient updating rejected application
            _setRecipientStatus(recipientId, uint8(Status.Appealed));
        }
        emit UpdatedRegistration(recipientId, _data, _sender,
↪   _getUintRecipientStatus(recipientId));
    }
}
```

`DonationVotingMerkleDistributionBaseStrategy._getUintRecipientStatus` calls `_getStatusRowColumn` to get the column index and current row. https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/donation-voting-merkle-base/DonationVotingMerkleDistributionBaseStrategy.sol#L819

```
function _getUintRecipientStatus(address _recipientId) internal view returns
↪   (uint8 status) {
    // Get the column index and current row
    (, uint256 colIndex, uint256 currentRow) = _getStatusRowColumn(_recipientId);

    // Get the status from the 'currentRow' shifting by the 'colIndex'
    status = uint8((currentRow >> colIndex) & 15);

    // Return the status
    return status;
}
```

`DonationVotingMerkleDistributionBaseStrategy._getStatusRowColumn` computes indexes from `recipientToStatusIndexes[_recipientId]`. For the new recipient. Those indexes should be zero. https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/donation-voting-merkle-base/DonationVotingMerkleDistributionBaseStrategy.sol#L833

```
function _getStatusRowColumn(address _recipientId) internal view returns
↪   (uint256, uint256, uint256) {
    uint256 recipientIndex = recipientToStatusIndexes[_recipientId];

    uint256 rowIndex = recipientIndex / 64; // 256 / 4
    uint256 colIndex = (recipientIndex % 64) * 4;

    return (rowIndex, colIndex, statusesBitMap[rowIndex]);
}
```

The problem is that `recipientCounter` starts from zero. https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/donation-voting-merkle-base/DonationVotingMerkleDistributionBaseStrategy.sol#L166

```
/// @notice The total number of recipients.
uint256 public recipientsCounter;
```

Consider the following situation:

- Alice is the first recipient calls `registerRecipient`

```
// in _registerRecipient
recipientToStatusIndexes[Alice] = recipientsCounter = 0;
_setRecipientStatus(Alice, uint8(Status.Pending));
recipientCounter++
```

- Bob calls `registerRecipient`.

```
// in _getStatusRowColumn
recipientToStatusIndexes[Bob] = 0 // It would access the status of Alice
// in _registerRecipient
currentStatus = _getUintRecipientStatus(recipientId) = Status.Pending
currentStatus != uint8(Status.None) -> no new application is recorded in the
↪  pool.
```

This implementation error makes the pool can only record the first application.

Impact

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/donation-voting-merkle-base/DonationVotingMerkleDistributionBaseStrategy.sol#L580 https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/donation-voting-merkle-base/DonationVotingMerkleDistributionBaseStrategy.sol#L819 https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/donation-voting-merkle-base/DonationVotingMerkleDistributionBaseStrategy.sol#L833 https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/donation-voting-merkle-base/DonationVotingMerkleDistributionBaseStrategy.sol#L166

Tool used

Manual Review

Recommendation

Make the counter start from 1. There are two methods to fix the issue.

1.

```
    /// @notice The total number of recipients.
+   uint256 public recipientsCounter;
-   uint256 public recipientsCounter;
```

2.

```
    function _registerRecipient(bytes memory _data, address _sender)
        internal
        override
        onlyActiveRegistration
        returns (address recipientId)
    {
        ...

        uint8 currentStatus = _getUintRecipientStatus(recipientId);

        if (currentStatus == uint8(Status.None)) {
            // recipient registering new application
+           recipientToStatusIndexes[recipientId] = recipientsCounter + 1;
-           recipientToStatusIndexes[recipientId] = recipientsCounter;
            _setRecipientStatus(recipientId, uint8(Status.Pending));

            bytes memory extendedData = abi.encode(_data, recipientsCounter);
            emit Registered(recipientId, extendedData, _sender);

            recipientsCounter++;
        ...
    }
```

Discussion

AhmadDecoded

Escalate

This should be upgraded to high, breaks the core functionality of protocol, setting wrong status.

sherlock-admin2

> Escalate
>
> This should be upgraded to high, breaks the core functionality of protocol, setting wrong status.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

kadenzipfel

To clarify impact, adding to @AhmadDecoded's escalation, since every new recipient after the initial registered recipient will have a `currentStatus != Status.None`, we will continually update the status of the initial recipient, never actually correctly marking any recipient statuses. Since the registration process is used to determine distribution, in the worst case, following the on-chain registration statuses as expected to determine distribution, the pool manager may incorrectly distribute the full poolAmount solely to the initial registered recipient. In the best case, the pool will not be usable and a new one will have to be redeployed, causing the pool manager and any registered recipients to lose a material amount of funds due to gas costs *every time* a pool is created with this strategy.

To summarize, since the impact is either (perhaps unlikely) significant fund loss by believing contract state to be true or (100% likely) less significant gas loss due to the system completely failing *every time*, this should be classified as high severity.

neeksec

Agree with Escalation that this is a high becasue of this impact described by @kadenzipfel,

> in the worst case, following the on-chain registration statuses as expected to determine distribution, the pool manager may incorrectly distribute the full poolAmount solely to the initial registered recipient.

MLON33

https://github.com/allo-protocol/allo-v2/pull/351

Evert0x

Planning to accept escalation and set severity to high

Evert0x

Result: High Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- ahmaddecoded: accepted

jkoppel

Why do the duplicates still have the Medium label?

jack-the-pug

Fixed. Note: `recipientsCounter` is now +1 than the actual count of recipients.

Issue H-3: `Registry.sol` generate clone `Anchor.sol` never work. Profile owner cannot use their `Anchor` wallet

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/380

Found by

0xG0P1, 0xnirlin, Arz, KingNFT, VAD37, niluk, pinalikefruit, zach030 User create new profile through `Registry`. Each profile have its own unique `Anchor` clone contract to handle transactions as a wallet. New clone `Anchor.sol` contract never work because `registry` address setup in `Anchor` constructor point to wrong address. This broke `Anchor` contract. Profile owner cannot use their wallet `Anchor`. All funds send to this `Anchor` contract will be lost forever.

Vulnerability Detail

Add this test to `Registry.t.sol` test file to reproduce the issue.

```
function test_Audit_createProfile() public {
    // create profile
    bytes32 newProfileId = registry().createProfile(nonce, name, metadata,
↪   profile1_owner(), profile1_members());
    Registry.Profile memory profile = registry().getProfileById(newProfileId);
    Anchor _anchor = Anchor(payable(profile.anchor));

    console.log("registry address: %s", address(registry()));
    console.log("anchor address: %s", profile.anchor);
    console.log("anchor.registry: %s", address(_anchor.registry()));

    emit log_named_bytes32("profile.id", profile.id);
    emit log_named_bytes32("anchor.profile.id", _anchor.profileId());

    Anchor _anchor_proxy = Anchor(payable(address( _anchor.registry())));
    assertEq(address(registry()),address(_anchor.registry()) ,"wrong anchor
↪   registry");
}
```

What happen with `Anchor.sol` is it expect `msg.sender` is `Registry` contract. But in reality `msg.sender` is a proxy contract generated by Solady during `CREATE3` operation.

```
constructor(bytes32 _profileId) {
    registry = Registry(msg.sender);//@audit H Registry address here is not
↪   Registry. msg.sender is a proxy contract. Create3 deploy 2 contract. one is
↪   proxy. other is actual bytecode.
    profileId = _profileId;
```

```
}
```

This can be seen with Solady comment for proxy contract. `msg.sender` above is middleman proxy contract. Not `Registry` contract. Solady generate 2 contract during CREATE3 operation. One is proxy contract. Second is actual bytecode.

Impact

`Anchor.execute()` function will not work because `registry` address point to empty proxy contract and not actual `Registry` so all call will revert.

```
File: allo-v2\contracts\core\Anchor.sol
70:     function execute(address _target, uint256 _value, bytes memory _data)
↪  external returns (bytes memory) {
71:         // Check if the caller is the owner of the profile and revert if not
72:         if (!registry.isOwnerOfProfile(profileId, msg.sender)) revert
↪  UNAUTHORIZED();
```

Profile owner cannot use their wallet `Anchor`. All funds send to this `Anchor` contract will be lost forever.

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/core/Registry.sol#L350 https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/core/Anchor.sol#L55-L58

Tool used

Manual Review

Recommendation

Move `msg.sender` into constructor parameter

```
File: allo-v2\contracts\core\Registry.sol
347:             bytes memory creationCode =
↪  abi.encodePacked(type(Anchor).creationCode, abi.encode(_profileId,
↪  address(this))); //@audit fix creation code
348:
349:             // Use CREATE3 to deploy the anchor contract
350:             anchor = CREATE3.deploy(salt, creationCode, 0);
File: allo-v2\contracts\core\Anchor.sol
55:     constructor(bytes32 _profileId, address _registry) {
56:         registry = Registry(_registry);
```

```
57:         profileId = _profileId;
58:     }
```

Discussion

jkoppel

Escalate.

This should be a medium because the error would be detected upon the first use of Anchor, with minimal funds lost. The owner would simply redeploy the Anchor and Registry contracts and call updateRegistry on Allo.

sherlock-admin2

> Escalate.
>
> This should be a medium because the error would be detected upon the first use of Anchor, with minimal funds lost. The owner would simply redeploy the Anchor and Registry contracts and call updateRegistry on Allo.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

0xArz

@jkoppel This should be a high because execute is not payable and the users will have to send ether to the Anchor which will then get stuck and this doesnt have to be detected upon the first use, many users can create an Anchor and deposit ether and only later call execute. A normal user cant just detect this issue if the execute reverts, he might think that its an error on his side. We dont know how fast this will be fixed, how many users will deposit and how much but its def not minimal loss of funds here.

jkoppel

I didn't say no funds would get stuck. I just said that little would.

AhmadDecoded

Assumption that little funds will be completely arbitrary.

neeksec

Suggest to keep high.

Agree with @0xArz and @AhmadDecoded's comments. The lost amount is not foreseeable and could be high.

MLON33

https://github.com/allo-protocol/allo-v2/pull/348

Evert0x

Planning to reject escalation and keep issue state as is.

Evert0x

Result: High Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- jkoppel: rejected

jack-the-pug

Fixed.

Issue H-4: Missing access modifier for `RFPSimpleStrategy.setPoolActive()` may lead to multiple issues

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/458

Found by

0x180db, 0xdeadbeef, 0xkaden, HChang26, HHK, Kow, fibonacci, jkoppel, lemonmon, rvierdiiev, simon135 `RFPSimpleStrategy.setPoolActive()` can be called by anybody since it's missing the `onlyPoolManager(msg.sender)` modifier, which can be abused by a malicious actor to steal funds.

Vulnerability Detail

The comment on line 217 in RFPSimpleStrategy.sol says that `'msg.sender' must be a pool manager` in order to be able to call `RFPSimpleStrategy.setPoolActive()`. However, the necessary `onlyPoolManager(msg.sender)` modifier is missing.

Impact

Multiple functions inside `RFPSimpleStrategy.sol` are either using the `onlyActivePool` or the `onlyInactivePool` modifiers:

- `RFPSimpleStrategy._distribute()`
- `RFPSimpleStrategy.withdraw()`
- `RFPSimpleStrategy._registerRecipient()`
- `RFPSimpleStrategy._allocate()`

A malicious actor (Alice) might do the following for example:

1. Alice registers themself as recipient for a `RFPSimpleStrategy`, specifying a `proposalBid` which is `15e18`.
2. Alice is being declared as the accepted recipient by the pool manager.
3. Now if the tokens were distributed to Alice, the amount of tokens Alice would receive would be `(15e18 * milestone.amountPercentage) / 1e18` (line 435 RFPSimpleStrategy.sol).
4. However, Alice calls `RFPSimpleStrategy.setPoolActive()` to make the pool active again, before the tokens are distributed. Alice might do this by either frontrunning or by executing the tx earlier.
5. Now Alice can call `RFPSimpleStrategy._registerRecipient()`, since the pool is active again, and Alice re-registers themself but with a higher `proposalBid`

than was accepted before (line 378 RFPSimpleStrategy.sol), for example they re-register with a `proposalBid` of `60e18`.

6. Then Alice calls `RFPSimpleStrategy.setPoolActive()` to set the pool inactive, so that the tokens can be distributed.
7. Now when the tokens are distributed to Alice for the first milestone (and later also for subsequent milestones), they receive a much higher amount of tokens, since Alice maliciously increased their accepted `proposalBid` from `15e18` to `60e18`, so they would now receive `(60e18 * milestone.amountPercentage) / 1e18` (line 435 RFPSimpleStrategy.sol) which is more than was accepted.

The above example illustrates how Alice can abuse setting the pool to active and inactive to change their accepted `proposalBid` to receive more tokens.

Also, Alice could potentially steal funds from the strategy, if they get accepted with a smaller `proposalBid` and then maliciously increase the `proposalBid` as described in the above example, so that Alice would receive a much higher amount of tokens that they are not eligible to receive and that are effectively being stolen from the funds of the strategy.

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/rfp-simple/RFPSimpleStrategy.sol#L217-L221

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/rfp-simple/RFPSimpleStrategy.sol#L417-L450

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/rfp-simple/RFPSimpleStrategy.sol#L314-L380

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/rfp-simple/RFPSimpleStrategy.sol#L386-L393

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/rfp-simple/RFPSimpleStrategy.sol#L295

Tool used

Manual Review

Recommendation

Consider adding the missing access modifier `onlyPoolManager(msg.sender)` to `RFPSimpleStrategy.setPoolActive()`.

Discussion

jkoppel

Escalate.

Nearly all the issues marked as duplicates of this issue are not duplicates. The vast majority identify that setPoolActive is missing access control but fail to identify any substantive consequences of this fact. In accordance with Sherlock rules, they should not be marked duplicates.

This was extensively discussed in the Discord. See, for instance, https://discord.com/channels/812037309376495636/1150807984893591643/1154681894261227551 .

sherlock-admin2

> Escalate.
>
> Nearly all the issues marked as duplicates of this issue are not duplicates. The vast majority identify that setPoolActive is missing access control but fail to identify any substantive consequences of this fact. In accordance with Sherlock rules, they should not be marked duplicates.
>
> This was extensively discussed in the Discord. See, for instance, https://discord.com/channels/812037309376495636/1150807984893591643/1154681894261227551 .

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

jkoppel

For instance:

I just reviewed the first 20 issues marked as duplicates of this one.

Specifically, I reviewed: #3 , #25 , #29 , #47 , #50 , #55 , #63 , #88 , #91 , #93 , #95 , #105 , #116 , #117 , #128, #130 , #134 , #135 , #143 , #147

Of those 20, **only 3 actually found a vulnerability**. They are #116 , #128 , and #130 . **Edit**: And #55. Who wants to help review the rest?

kadenzipfel

To add to @jkoppel's escalation, this vulnerability specifically references the ability to frontrun `_distribute` with a re-registration, allowing the attacker to steal funds. The lack of authorization validation in `setPoolActive` is simply a dependency of the vulnerability. It's imperative that we separate "lack of `setPoolActive` authorization" submissions based on the actual impact. The other impact present from this lack of

authorizations is griefing as defined in https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/356 and should be classified separately.

rvierdiyev

I am not going to take any side in this escalation, however in order to protect myself from possible results will say, that my issue #55 also stated about ability to change bid amount with frontrunning

> Another usage of this will be when distribute for milestone will be called to frontrun it to set pool to active, then change recipient bid to bigger amount, then again set pool to inactive and receive bigger payment.

0x00ffDa

I agree with jkoppel and kadenzipfel.

> The other impact present from this lack of authorizations is griefing as defined in #356 and should be classified separately.

If this ends up getting split according to attack method, #574 is also a griefing / DoS attack but more specific.

jkoppel

The griefing attacks do not work. They require front-running every transaction, and, if it starts getting annoying, users can just start prefixing their calls with setPoolActive().

Abelaby

Following @rvierdiyev example I want to drop in my issue #597 I have stated multiple potential issues that could arise from this lack of access modifier including the frontrun issue.

jkoppel

@Abelaby Where do you mention the use of frontrunning to steal funds? I only see the griefing attack in your writeup.

Abelaby

@jkoppel oh mb, I was referring to the DoS caused by front running in my case.

On another note, referring to your comment above;

> The griefing attacks do not work. They require front-running every transaction, and, if it starts getting annoying, users can just start prefixing their calls with setPoolActive().

I believe griefing attacks are very much possible, the malicious user can just listen for specific transactions and front run them with setPoolActive(). Even if gets annoying, the 'average' user might just not be aware of or will start prefixing their calls with setPoolActive().

And hypothetically, if the attacker is dedicated enough, their script can be smart enough to just not `frontrun` transactions with `prefixing` `setPoolActive()` so that the users prefixing their calls with `setPoolActive()` will just sabotage themselves.

And frontrunning every transactions are very much possible in chains with low cost, making this a valid attack vector.

jkoppel

I remain in favor of classifying the griefing attacks as a Low, but have nothing factual to add.

I've gone through the same set of 20 reports. In addition to the 4 that mention the High attack, another 2 mention the griefing attack: #63 and #88. There is an argument that #95 mentions it, but you have to squint hard.

If the griefing attack is classified as a Medium, then Sherlock rules state that such reports should be kept as a duplicate of this one.

> Scenario A: There is a root cause/error/vulnerability A in the code. This vulnerability A -> leads to two attack paths:
>
> - B -> high severity path
> - C -> medium severity attack path/just identifying the vulnerability. Both B & C would not have been possible if error A did not exist in the first place. In this case, both B & C should be put together as duplicates.
> - In addition to this, there is a submission D which identifies the core issue but does not clearly describe the impact or an attack path. Then D is considered low.

neeksec

Agree with Escalation.

The Sherlock docs are clear in this senario.

> Scenario A: There is a root cause/error/vulnerability A in the code. This vulnerability A -> leads to two attack paths:
>
> - B -> high severity path
> - C -> medium severity attack path/just identifying the vulnerability. Both B & C would not have been possible if error A did not exist in the first place. In this case, both B & C should be put together as duplicates.
> - In addition to this, there is a submission D which identifies the core issue but does not clearly describe the impact or an attack path. Then D is considered low. Scenario B: In the above example if the root issue A is one of the following generic vulnerabilities:

- Reentrancy
- Access control
- Front-running Then the submissions with valid attack paths and higher vulnerability are considered valid. If the submission is vague or does not identify the attack path with higher severity clearly it will be considered low.
- B is a valid issue
- C is low

DoS/griefing is low in this case.

Valid duplicates, #055, #116, #128, #130, #151, #188, #246, #255, #320, #450

Other submissions should be put low severity.

Comment if I mis-picked or missed any.

osmanozdemir1

If the thing that matters is not the bug itself (`setPoolActive` being external) but the impact of the bug and how the bug can be used (front-running and increasing the `proposalBid`), then issues should be duped according to these impact not the bug itself.

All of the issues mentioned above describes front-running and increasing the `proposalBid`. That's the impact of the bug. That's why only these issues rewarded, right? Only validating these issues means the actual problem is not `setPoolActive` being external, it is the `proposalBid` being front-runned. But that is already a confirmed issue. #497

If the important thing is the impact, it doesn't matter which function is being front-runned. `_allocate` or `_distribute` or `_registerRecipient`. What's the difference? The actual attack is front-running and increasing the `proposalBid`. If only these 11 issues going to be validated, they should be duped with #497

jkoppel

@osmanozdemir1

The Sherlock docs are very clear about this. To qualify, a report must find both the root cause and any Medium or High attack path, and then the reports are grouped by root cause.

Yes, there is a second attack that also involves front-running that transaction, but it works differently. A set of transactions that exploits this issue will not exploit that issue, nor vice versa. A fix to this issue will not fix that one, nor vice versa.

Not in Sherlock docs, but a question I like to ask when thinking about dups: if the sponsor had only gotten this report, and was committed to only fixing medium and

high vulnerabilities, would that be sufficient? If they had only gotten #497 and not this or its dupes, then the issue would be at large in the code.

jacksanford1

https://github.com/allo-protocol/allo-v2/pull/340

osmanozdemir1

> @osmanozdemir1
>
> The Sherlock docs are very clear about this. To qualify, a report must find both the root cause and any Medium or High attack path, and then the reports are grouped by root cause.
>
> Yes, there is a second attack that also involves front-running that transaction, but it works differently. A set of transactions that exploits this issue will not exploit that issue, nor vice versa. A fix to this issue will not fix that one, nor vice versa.
>
> Not in Sherlock docs, but a question I like to ask when thinking about dups: if the sponsor had only gotten this report, and was committed to only fixing medium and high vulnerabilities, would that be sufficient? If they had only gotten #497 and not this or its dupes, then the issue would be at large in the code.

A similar question can be ask this way: If the protocol had only gotten a few `setPoolActive` reports, but none of these reports mentioned any high/medium scenario, wouldn't the protocol fix this issue? They would definitely fix it because the bug itself is clear like a blue sky.

But it's okay, I get your point and rules are the rules. It is a good lesson for me and for some others :)

Evert0x

Planning to accept escalation and remove all duplicates except

https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/55, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/116, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/128, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/130, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/151, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/188, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/246, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/255, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/320, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/450

Evert0x

Result: High Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- jkoppel: accepted

jack-the-pug

Fixed.

Issue H-5: Malicious registrant can front-run `RFPSimpleStrategy._allocate()` in order to change the `proposalBid` and get a bigger payout in the distribution

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/497

Found by

0xbepresent, 0xdeadbeef, WATCHPUG, detectiveking, honeymewn, jah, jkoppel, qbs, rvierdiiev The `RFPSimpleStrategy::_allocate()` function can be frontrun by a malicious `registrant` chainging the `proposalBid` and get a bigger payout in the `RFPSimpleStrategy::_distribute()` function.

Vulnerability Detail

Users can register to the pool strategy using the RFPSimpleStrategy::_registerRecipient() function specifying the proposalBid in the registration. Then the pool manager accepts the registrant recipient using the RFPSimpleStrategy::_allocate() function.

The problem is that the execution of the `RFPSimpleStrategy::_allocate()` function by the `pool manager` can be frontun by a malicious `registrant recipient`. Consider the next scenario:

1. `UserA` call the `RFPSimpleStrategy::_registerRecipient()` using a `proposalBid=10`.
2. Pool manager accepts the proposal by `UserA` and call the `RFPSimpleStrategy::_allocate()` function.
3. `UserA` monitors the mempool and frontrun the manager `_allocate()` execution changing the proposal now `proposalBid=50`.
4. The `step 2` call finally is executed but the using non-agreed proposal `proposalBid=50`.

Now the `UserA` is accepted registrant recipient with non-agreed proposal bid (`proposalBid=50`).

Impact

Malicious registrant can change the `proposalBid` to a non-agreed term causing that he can receive a bigger payout in the RFPSimpleStrategy::_distribute() function because in the code line 435 the `proposalBid` is used to calculate the amount to pay to the `accepted registrant recipient`:

```
File: RFPSimpleStrategy.sol
417:     function _distribute(address[] memory, bytes memory, address _sender)
418:         internal
419:         virtual
420:         override
421:         onlyInactivePool
422:         onlyPoolManager(_sender)
423:     {
...
...
433:
434:         // Calculate the amount to be distributed for the milestone
435:         uint256 amount = (recipient.proposalBid *
↪  milestone.amountPercentage) / 1e18;
436:
437:         // Get the pool, subtract the amount and transfer to the recipient
438:         poolAmount -= amount;
439:         _transferAmount(pool.token, recipient.recipientAddress, amount);
...
...
450:     }
```

The malicious accepted registrant can drain all funds from the pool strategy using one milestone.

Code Snippet

- RFPSimpleStrategy::_registerRecipient()
- RFPSimpleStrategy::_allocate()
- RFPSimpleStrategy::_distribute()

Tool used

Manual review

Recommendation

Verify the `proposalBid` when the `_allocate()` occurs:

```
    function _allocate(bytes memory _data, address _sender)
        internal
        virtual
        override
        nonReentrant
```

```
        onlyActivePool
        onlyPoolManager(_sender)
    {
        // Decode the '_data'
--      acceptedRecipientId = abi.decode(_data, (address));
++      (acceptedRecipientId, uint256 expectedProposalBid) = abi.decode(_data,
 ↪  (address, uint256));

        Recipient storage recipient = _recipients[acceptedRecipientId];

--      if (acceptedRecipientId == address(0) || recipient.recipientStatus !=
 ↪  Status.Pending) {
++      if (acceptedRecipientId == address(0) || recipient.recipientStatus !=
 ↪  Status.Pending || recipient.proposalBid != expectedProposalBid) {
            revert RECIPIENT_ERROR(acceptedRecipientId);
        }

        // Update status of acceptedRecipientId to accepted
        recipient.recipientStatus = Status.Accepted;

        _setPoolActive(false);

        IAllo.Pool memory pool = allo.getPool(poolId);

        // Emit event for the allocation
        emit Allocated(acceptedRecipientId, recipient.proposalBid, pool.token,
 ↪  _sender);
    }
```

Discussion

thelostone-mc

Yup this does feel safer. The only downside is that the pool manager could now put in a big lower than what was proposed by the recipient but this seems alright as the pool manager is a trusted actor

kadenzipfel

Escalate This is simply a less severe version of frontrunning `_distribute` by flipping the unprotected `setPoolActive` flag, as described in https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/458.

There are two directions which may be reasonable in reclassifying this:

1) Reduce the severity to medium since the pool manager can become aware of this attack vector and not fulfill distributions for attackers (note that there is a `withdraw` function and thus not all funds must be distributed). This is inline

with the medium severity classifications as, "There is a viable scenario (even if unlikely) that could cause the protocol to enter a state where a material amount of funds can be lost." Whereas for https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/458, the high severity classification of, "This vulnerability *would* result in a material loss of funds" is true because the pool manager cannot prevent distribution after already submitting the transaction.

2) If however, this vulnerability is maintained as high severity (which I would disagree with), it should be duplicated with https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/458 as they are both fundamentally the same attack vector, although this one is less significant.

sherlock-admin2

> Escalate This is simply a less severe version of frontrunning `_distribute` by flipping the unprotected `setPoolActive` flag, as described in https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/458.
>
> There are two directions which may be reasonable in reclassifying this:
>
> 1) Reduce the severity to medium since the pool manager can become aware of this attack vector and not fulfill distributions for attackers (note that there is a `withdraw` function and thus not all funds must be distributed). This is inline with the medium severity classifications as, "There is a viable scenario (even if unlikely) that could cause the protocol to enter a state where a material amount of funds can be lost." Whereas for https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/458, the high severity classification of, "This vulnerability *would* result in a material loss of funds" is true because the pool manager cannot prevent distribution after already submitting the transaction.
> 2) If however, this vulnerability is maintained as high severity (which I would disagree with), it should be duplicated with https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/458 as they are both fundamentally the same attack vector, although this one is less significant.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

jkoppel

Similar attack vector, different issue. Fixing #458 does not fix this. The proposed fix of this issue has no impact on #458. See #378 for a way to use this attack which is far harder to defend against.

neeksec

Side with @jkoppel and sugguest to keep the orginal judging.

jacksanford1

https://github.com/allo-protocol/allo-v2/pull/346

Evert0x

Planning to reject escalation and keep issue state as is.

Evert0x

Result: High Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- kadenzipfel: rejected

jack-the-pug

Fixed.

A better fix would be to make it impossible to change the proposal once it is in the allocation stage:
https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/657

Issue M-1: `fundPool` does not work with fee-on-transfer token

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/19

Found by

0x1337, 0x180db, 0x6980, 0xHelium, 0xMosh, 0xbepresent, 0xdeadbeef, 0xgoat, Aamirusmani1552, ArmedGoose, AsenXDeth, BenRai, DevABDee, Inspex, JP_Courses, Kodyvim, Kow, Martians, Proxy, Tri-pathi, Vagner, WATCHPUG, ace13567, adeolu, alexzoid, ashirleyshe, ast3ros, cats, detectiveking, foresthalberd, grearlake, imsrybr0, inspecktor, inzinko, lealCodes, lemonmon, lil.eth, marchev, nobody2018, osmanozdemir1, p0wd3r, parsely, pavankv241, pengun, pontifex, qbs, rvierdiiev, seeques, shtesesamoubiq, theclonedtyroneidgafmf, trevorjudice, tsvetanovv, vagrant, xAriextz

Vulnerability Detail

In `_fundPool`, the parameter for `increasePoolAmount` is directly the amount used in the `transferFrom` call.

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/core/Allo.sol#L516-L517

```
_transferAmountFrom(_token, TransferData({from: msg.sender, to:
↪  address(_strategy), amount: amountAfterFee}));
_strategy.increasePoolAmount(amountAfterFee);
```

When `_token` is a fee-on-transfer token, the actual amount transferred to `_strategy` will be less than `amountAfterFee`. Therefore, the current approach could lead to a recorded balance that is greater than the actual balance.

Impact

`fundPool` does not work with fee-on-transfer token

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/core/Allo.sol#L516-L517

Tool used

Manual Review

Recommendation

Use the change in `_token` balance as the parameter for `increasePoolAmount`.

Discussion

MLON33

https://github.com/allo-protocol/allo-v2/pull/355

quentin-abei

Should consider choosing this issue for report : 30 It's better detailed and have an actual working coded PoC

jack-the-pug

Fixed.

Issue M-2: Exponential Inflation of Voice Credits in Quadratic Voting Strategy

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/48

Found by

0x00ffDa, 0x3b, 0xRstStn, 0xarno, 0xbepresent, 0xc0ffEE, 0xdeadbeef, 0xkaden, 0xpoolbaer, Arz, AsenXDeth, BenRai, GimelSec, HChang26, HHK, KingNFT, Kow, Topmark, WATCHPUG, ZdravkoHr., ace13567, alexxander, ashirleyshe, ast3ros, chaduke, coffiasd, dany.armstrong90, detectiveking, honeymewn, jah, jkoppel, jovi, lemonmon, lil.eth, nobody2018, osmanozdemir1, pengun, pontifex, rvierdiiev, seeques, shtesesamoubiq, simon135, tnquanghuy0512, toshii, trachev, twcctop

Vulnerability Detail

In the given code snippet, we observe a potential issue in the way voice credits are being accumulated for each recipient. The specific lines of code in question are:

```
function _qv_allocate(
        ...
    ) internal onlyActiveAllocation {
        ...
        uint256 creditsCastToRecipient =
↪   _allocator.voiceCreditsCastToRecipient[_recipientId];
        ...
        // get the total credits and calculate the vote result
        uint256 totalCredits = _voiceCreditsToAllocate + creditsCastToRecipient;
        ...
        //E update allocator mapping voice for this recipient
        _allocator.voiceCreditsCastToRecipient[_recipientId] += totalCredits;
↪   //E @question should be only _voiceCreditsToAllocate
        ...
    }
```

We can see that at the end :

```
_allocator.voiceCreditsCastToRecipient[_recipientId] =
↪   _allocator.voiceCreditsCastToRecipient[_recipientId] +
↪   _voiceCreditsToAllocate +
↪   _allocator.voiceCreditsCastToRecipient[_recipientId];
```

Here, totalCredits accumulates both the newly allocated voice credits (`_voiceCreditsToAllocate`) and the credits previously cast to this recipient (`creditsCastToRecipient`). Later on, this totalCredits is added again to

`voiceCreditsCastToRecipient[_recipientId]`, thereby including the previously cast credits once more

Proof of Concept (POC):

Let's consider a scenario where a user allocates credits in three separate transactions:

1. Transaction 1: Allocates 5 credits
- creditsCastToRecipient initially is 0
- totalCredits = 5 (5 + 0)
- New voiceCreditsCastToRecipient[_recipientId] = 5
2. Transaction 2: Allocates another 5 credits
- creditsCastToRecipient now is 5 (from previous transaction)
- totalCredits = 10 (5 + 5)
- New voiceCreditsCastToRecipient[_recipientId] = 15 (10 + 5)
3. Transaction 3: Allocates another 5 credits
- creditsCastToRecipient now is 15
- totalCredits = 20 (5 + 15)
- New voiceCreditsCastToRecipient[_recipientId] = 35 (20 + 15)

From the above, we can see that the voice credits cast to the recipient are exponentially growing with each transaction instead of linearly increasing by 5 each time

Impact

Exponential increase in the voice credits attributed to a recipient, significantly skewing the results of the voting strategy(if one recipient receive 15 votes in one vote and another one receive 5 votes 3 times, the second one will have 20 votes and the first one 15) Over time, this could allow for manipulation and loss of trust in the voting mechanism and the percentage of amount received by recipients as long as allocations are used to calculate the match amount they will receive from the pool amount.

Code Snippet

https://github.com/allo-protocol/allo-v2/blob/main/contracts/strategies/qv-base/QVBaseStrategy.sol#L529

Tool used

Manual Review

Recommendation

Code should be modified to only add the new voice credits to the recipient's tally. The modified line of code should look like:

```
_allocator.voiceCreditsCastToRecipient[_recipientId] += _voiceCreditsToAllocate;
```

Discussion

MLON33

https://github.com/allo-protocol/allo-v2/pull/338

jack-the-pug

Fixed

Issue M-3: RFPSimpleStrategy milestones can be set multiple times

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/176

Found by

GimelSec, HChang26, Martians, WATCHPUG, ace13567, fibonacci, jkoppel, lemonmon, osmanozdemir1 Until the first distribution is completed, it's possible to call `setMilestones` function multiple times. New milestones are added to the previous ones. The `totalAmountPercentage` of all milestones in this case will be greater than 100%. It also affects all the contracts that are inherited from RFPSimpleStrategy.

Vulnerability Detail

The `setMilestones` function in `RFPSimpleStrategy` contract checks if `MILESTONES_ALREADY_SET` or not by `upcomingMilestone` index.

```
if (upcomingMilestone != 0) revert MILESTONES_ALREADY_SET();
```

But `upcomingMilestone` increases only after distribution, and until this time will always be equal to 0.

Impact

It can accidentally break the pool state or be used with malicious intentions.

1. Two managers accidentally set the same milestones. Milestones are duplicated and can't be reset, the pool needs to be recreated.
2. The manager, in cahoots with the recipient, sets milestones one by one, thereby bypassing `totalAmountPercentage` check and increasing the payout amount.

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/rfp-simple/RFPSimpleStrategy.sol#L224-L247

Tool used

Manual Review

Recommendation

Fix condition if milestones should only be set once.

```
if (milestones.length > 0) revert MILESTONES_ALREADY_SET();
```

Or allow milestones to be reset while they are not in use.

```
if (milestones.length > 0) {
    if (milestones[0].milestoneStatus != Status.None) revert
↪   MILESTONES_ALREADY_IN_USE();
    delete milestones;
}
```

Discussion

sherlock-admin2

> Escalate This is invalid. I cannot accept that a pool manager can be malicious

```
You've deleted an escalation for this issue.
```

jkoppel

#376 explains how this can cause an issue without a malicious pool owner.

MLON33

https://github.com/allo-protocol/allo-v2/pull/341

jack-the-pug

Fixed.

Issue M-4: Allo#_fundPool

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/198

Found by

0xkaden, AsenXDeth, HChang26, Martians, RadCet, SBSecurity, Shubham, alexxander, imare, sashik_eth, trachev, vangrim

Vulnerability Detail

Let's see the code of the `_fundPool` function:

```
function _fundPool(uint256 _amount, uint256 _poolId, IStrategy _strategy)
↪  internal {
        uint256 feeAmount;
        uint256 amountAfterFee = _amount;

        Pool storage pool = pools[_poolId];
        address _token = pool.token;

        if (percentFee > 0) {
            feeAmount = (_amount * percentFee) / getFeeDenominator();
            amountAfterFee -= feeAmount;

            _transferAmountFrom(_token, TransferData({from: msg.sender, to:
↪  treasury, amount: feeAmount}));
        }

        _transferAmountFrom(_token, TransferData({from: msg.sender, to:
↪  address(_strategy), amount: amountAfterFee}));
        _strategy.increasePoolAmount(amountAfterFee);

        emit PoolFunded(_poolId, amountAfterFee, feeAmount);
    }
```

The `feeAmount` is calculated as follows:

```
feeAmount = (_amount * percentFee) / getFeeDenominator();
```

where `getFeeDenominator` returns `1e18` and `percentFee` is represented like that: 1e18 = 100%, 1e17 = 10%, 1e16 = 1%, 1e15 = 0.1% (from the comments when declaring the variable).

Let's say the pool uses a token like GeminiUSD which is a token with 300M+ market cap, so it's widely used, and `percentFee` == 1e15 (0.1%)

A user could circumvent the fee by depositing a relatively small amount. In our example, he can deposit 9 GeminiUSD. In that case, the calculation will be: `feeAmount = (_amount * percentFee) / getFeeDenominator() = (9e2 * 1e15) / 1e18 = 9e17/1e18 = 9/10 = 0;`

So the user ends up paying no fee. There is nothing stopping the user from funding his pool by invoking the `fundPool` with such a small amount as many times as he needs to fund the pool with whatever amount he chooses, circumventing the fee.

Especially with the low gas fees on L2s on which the protocol will be deployed, this will be a viable method to fund a pool without paying any fee to the protocol.

Impact

The protocol doesn't collect fees from pools with low decimal tokens.

Code Snippet

https://github.com/allo-protocol/allo-v2/blob/main/contracts/core/Allo.sol#L502

Tool used

Manual Review

Recommendation

Add a `minFundAmount` variable and check for it when funding a pool.

Discussion

sherlock-admin

1 comment(s) were left on this issue during the judging contest.

n33k commented:

> bypassing of fee is an acceptable risk

nevillehuang

Escalate

In the contest README here:

> Fee skirting where pool manager directly fund the pool without paying the fees

Fee bypass is only acceptable for pool managers funding pools, NOT just anybody. Some form of check needs to be implemented to check if `msg.sender` is a pool manager to allow fee skirting (if it even is intentional).

Sidenote: #248, #253, #533, #571, #602, #628, #636, #706 ,#764, #812, #818, #825, #911, #947 are valid duplicates of this issue

sherlock-admin2

> Escalate
>
> In the contest README here:
>
> > Fee skirting where pool manager directly fund the pool without paying the fees
>
> Fee bypass is only acceptable for pool managers funding pools, NOT just anybody. Some form of check needs to be implemented to check if `msg.sender` is a pool manager to allow fee skirting (if it even is intentional).
>
> Sidenote: #248, #253, #533, #571, #602, #628, #636, #706 ,#764, #812, #818, #825, #911, #947 are valid duplicates of this issue

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

neeksec

Suggest to keep the original judging.

> > Fee skirting where pool manager directly fund the pool without paying the fees
>
> Fee bypass is only acceptable for pool managers funding pools, NOT just anybody.

It's acceptable for anybody. If it's not, pool managers can use another EOA to do the fee skirting which easily voids this contest rule.

nevillehuang

While i respect your point, based on contest READ.ME, only pool managers can bypass fees. Unless it is intended protocol functionality to allow anyone other than pool manager to bypass fees, some sort of check is required. (I.e. minimum fee amount check)

Evert0x

Planning to accept escalation make issue medium together with listed duplicates.

> It's acceptable for anybody

It's clear that it is NOT an acceptable risk for anybody. It's only an acceptable risk for the pool manager role.

> **Please list any known issues/acceptable risks that should not result in a valid finding.** Fee skirting where pool manager directly fund the pool without paying the fees

Says pool manager, not anybody.

Evert0x

Result: Medium Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- nevillehuang: accepted

Issue M-5: The `RFPSimpleStrategy._registerRecipient()` does not work when the strategy was created using the `useRegistryAnchor=true` causing that nobody can register to the pool

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/245

Found by

0xMAKEOUTHILL, 0xarno, 0xbepresent, 0xnirlin, GimelSec, HHK, KingNFT, Kow, Martians, Nyx, SBSecurity, Vagner, WATCHPUG, ace13567, alexxander, ashirleyshe, ast3ros, fibonacci, fishgang, hindsight, jah, jkoppel, nobody2018, osmanozdemir1, pontifex, rvierdiiev, sashik_eth, tnquanghuy0512, toshii, unix515, ustas, xAriextz, zach030 The RFPSimpleStrategy._registerRecipient() does not work when the strategy was created using the `useRegistryAnchor=true` causing that no one can register to the pool and the funds sent to the pool may be trapped.

Vulnerability Detail

The `RFPSimpleStrategy` strategies can be created using the `useRegistryAnchor` which indicates whether to use the registry anchor or not. If the pool is created using the `useRegistryAnchor=true` the RFPSimpleStrategy._registerRecipient() will be reverted by RECIPIENT_ERROR. The problem is that when `useRegistryAnchor` is true, the variable recipientAddress is not collected so the function will revert by the RECIPIENT_ERROR.

I created a test where the strategy is created using the `userRegistryAnchor=true` then the `registerRecipient()` will be reverted by the `RECIPIENT_ERROR`.

```
// File: test/foundry/strategies/RFPSimpleStrategy.t.sol:RFPSimpleStrategyTest
// $ forge test --match-test
↪  "test_registrationIsBlockedWhenThePoolIsCreatedWithUseRegistryIsTrue" -vvv
//
    function
↪  test_registrationIsBlockedWhenThePoolIsCreatedWithUseRegistryIsTrue() public
↪  {
        // The registerRecipient() function does not work then the strategy was
↪  created using the
        // useRegistryAnchor = true.
        //
        bool useRegistryAnchorTrue = true;
        RFPSimpleStrategy custom_strategy = new
↪  RFPSimpleStrategy(address(allo()), "RFPSimpleStrategy");

        vm.prank(pool_admin());
```

```
        poolId = allo().createPoolWithCustomStrategy(
            poolProfile_id(),
            address(custom_strategy),
            abi.encode(maxBid, useRegistryAnchorTrue, metadataRequired),
            NATIVE,
            0,
            poolMetadata,
            pool_managers()
        );
        //
        // Create profile1 metadata and anchor
        Metadata memory metadata = Metadata({protocol: 1, pointer: "metadata"});
        address anchor = profile1_anchor();
        bytes memory data = abi.encode(anchor, 1e18, metadata);
        //
        // Profile1 member registers to the pool but it reverted by
↪   RECIPIENT_ERROR
        vm.startPrank(address(profile1_member1()));
        vm.expectRevert(abi.encodeWithSelector(RECIPIENT_ERROR.selector,
↪   address(anchor)));
        allo().registerRecipient(poolId, data);
    }
```

Impact

The pool created with a strategy using the `userRegistryAnchor=true` can not get registrants because `_registerRecipient()` will be reverted all the time. If the pool is funded but no one can be allocated since there is not registered recipients, the deposited funds by others may be trapped because those are not distributed since there are not `registrants`.

Code Snippet

- _registerRecipient()

Tool used

Manual review

Recommendation

When the strategy is using `useRegistryAncho=true`, get the `recipientAddress` from the `data`:

```
    function _registerRecipient(bytes memory _data, address _sender)
        internal
```


```
        override
        onlyActivePool
        returns (address recipientId)
    {
        bool isUsingRegistryAnchor;
        address recipientAddress;
        address registryAnchor;
        uint256 proposalBid;
        Metadata memory metadata;

        // Decode '_data' depending on the 'useRegistryAnchor' flag
        if (useRegistryAnchor) {
            /// @custom:data when 'true' -> (address recipientId, uint256
↪   proposalBid, Metadata metadata)
--          (recipientId, proposalBid, metadata) = abi.decode(_data, (address,
↪   uint256, Metadata));
++          (recipientId, recipientAddress, proposalBid, metadata) =
↪   abi.decode(_data, (address, address, uint256, Metadata));

            // If the sender is not a profile member this will revert
            if (!_isProfileMember(recipientId, _sender)) revert UNAUTHORIZED();
```

Discussion

MLON33

https://github.com/allo-protocol/allo-v2/pull/342

jack-the-pug

Fixed.

Issue M-6: QV strategy cannot receive native token

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/256

Found by

0xarno, 0xc0ffEE, 0xkaden, Arz, B353N, Martians, Proxy, WATCHPUG, alexxander, bronze_pickaxe, jkoppel, tnquanghuy0512, twcctop, vangrim The RFP and DonationMerkle strategies have a receive() method so that they can be funded with native token. The QV strategy does not, and therefore it is incompatible with the native token.

Vulnerability Detail

See summary

Impact

Cannot use QV strategy with native token.

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-base/QVBaseStrategy.sol#L574

Notice the lack of a receive() method, which is also lacking in QVSimpleStrategy. In contrast, the others have it, e.g.: https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/rfp-simple/RFPSimpleStrategy.sol#L500 .

Allo.fundPool calls _transferAmountFrom: https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/core/Allo.sol#L516 . Tracing the definitions, it invokes the strategy with `call(gas(), to, amount, gas(), 0x00, gas(), 0x00)` (in SafeTransferLib). I.e.: it invokes it with empty calldata, and will therefore try to call receive() and revert.

Tool used

Manual Review

Recommendation

Add a receive() method.

Discussion

sherlock-admin

1 comment(s) were left on this issue during the judging contest.

n33k commented:

> low, no fund lose, only incompatible with NATIVE token, should consider to fix

jkoppel

Escalate.

Actually, there can be fund loss. If you attempt to create a pool but do not grant it initial funds, then you'll have paid the baseFee for pool creation but will have an inoperable pool. Further, Sherlock rules do consider valid issues that prevent contracts from achieving their purpose, even in the absence of fund loss.

sherlock-admin2

> Escalate.
>
> Actually, there can be fund loss. If you attempt to create a pool but do not grant it initial funds, then you'll have paid the baseFee for pool creation but will have an inoperable pool. Further, Sherlock rules do consider valid issues that prevent contracts from achieving their purpose, even in the absence of fund loss.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

> Escalate.
>
> Actually, there can be fund loss. If you attempt to create a pool but do not grant it initial funds, then you'll have paid the baseFee for pool creation but will have an inoperable pool. Further, Sherlock rules do consider valid issues that prevent contracts from achieving their purpose, even in the absence of fund loss.

If it reverts due to lack of receive payable, doesn't it mean pool creator get back the fees they paid?

But i do agree that this breaks core functionality since it is explicitly stated by protocol that all tokens are supported, even native ETH which is so widely used. I am inclined to think this is medium severity since i am very sure the protocol does

not intend to not support native ETH for funding, which can potentially cause loss of fees.

Sidenote: #27, #299, #363, #506, #521, #541, #549, #595, #624, #677, #894, #941, #957 are valid duplicates of this issue.

jkoppel

Nice work finding the duplicates!

> If it reverts due to lack of receive payable, doesn't it mean pool creator get back the fees they paid?

Only if you try to fund the pool at creation time.

0x00ffDa

To qualify for Medium, I think it depends whether the fees lost would constitute "a material amount of funds". Also, the protocol owner can refund the fee to the user, so arguably not lost at all.

For me, I didn't report this because, although it's a valid issue ... I expected it to be judged Low. It will be interesting to see what they decide.

thelostone-mc

Would agree that this is an issue which needs to be fixed

neeksec

Suggest to keep make this low/info.

The fees lost is not "a material amount of funds" and could be refunded by Allo team.

MLON33

https://github.com/allo-protocol/allo-v2/pull/376

Evert0x

I believe this falls into the same category as https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/862#issuecomment-1777391236

https://docs.sherlock.xyz/audits/judging/judging#v.-how-to-identify-a-medium-issue

> Breaks core contract functionality, rendering the contract useless

Planning to accept and make medium, but will get back to this.

Evert0x

> But i do agree that this breaks core functionality since it is explicitly stated by protocol that all tokens are supported, even native ETH

@nevillehuang where is this stated?

nevillehuang

> But i do agree that this breaks core functionality since it is explicitly stated by protocol that all tokens are supported, even native ETH
>
> @nevillehuang where is this stated?

In their docs here, which is part of the contest details.

> Pools can also be funded with either native Ether or an ERC20 token. Note though that a pool can only distribute one token, which is determined when the pool is created. Trying to call fundPool with a token other than the one used when the pool was created will cause the method to revert with an error. Pools should always be funded using the fundPool method and you should never transfer funds directly to Allo.sol

Evert0x

Based on the language in their docs it's clear that receiving native Ether is a core functionality of the protocol.

Still planning to go with Medium with https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/27, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/299, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/363, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/506, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/521, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/541, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/549, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/595, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/624, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/677, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/894, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/941, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/957 as dupes

Evert0x

Result: Medium Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- jkoppel: accepted

Issue M-7: The 1-second overlap between the during- and after-allocation periods may cause funds to become stuck, permanently.

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/261

Found by

alexxander, jkoppel, shirochan There is a 1-second overlap between the during- and after-allocation periods of both the QV strategy and the DonationVotingMerkle strategies. In the QV strategy, this can lead to funds permanently becoming stuck.

Vulnerability Detail

1. A QVSimpleStrategy pool is created. Alice is the Pool manager. 10 people are given voting rights. The pool is funded for 100k USDC.
2. Everyone except Bob votes, giving Carol and Dan each 50% of the votes.
3. Carol tells Alice she really needs the money, and asks her to distribute ASAP. Alice schedules a distribute() transaction to occur the moment allocation ends.
4. At the last minute, the Bob votes, casting all his votes for Carol.
5. It so happens that Alice and Bob's transaction occur in the same block. Further, it so happens that this block is scheduled with block.timestamp exactly equal to `allocationEndTime`. (Note that, other some comments talk about the registration and allocation period times being in milliseconds, they are actually in seconds.)
6. For this second, both `onlyActiveAllocation` and `onlyAfterAllocation` pass
7. Bob's vote gets scheduled within the block after the call to distribute()
8. The call to distribute gives 50% of the funds to Carol, or 50k USDC. Bob's vote changes the total so that 55% of votes are for Carol, and 45% for Dan.
9. Dan can now be given his $45k. But the remaining $5k is stuck in the pool forever.

If there is one block every 40 seconds, then there is a 2.5% chance that a block will run on the exact second of overlap. However, the contest page says it should run on any EVM-compatible chain. If there is a rollup that has blocks every second, then the conditions for this bug to occur has a 100% chance

If there is collusion from the miner, then there is also a significantly higher chance.

Impact

A chance of permanent loss of funds

Code Snippet

When `block.timestamp == allocationEndTime`, then both _checkOnlyAfterRegistration and _checkOnlyActiveAllocation pass.

I am guessing that the protocol authors thought `block.timestamp` was in milliseconds, when it is actually in seconds. That makes this problem 1000x more likely to occur without miner collusion.

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-base/QVBaseStrategy.sol#L317C1-L328C6

```
function _checkOnlyActiveAllocation() internal view virtual {
    if (allocationStartTime > block.timestamp || block.timestamp >
↪   allocationEndTime) {
        revert ALLOCATION_NOT_ACTIVE();
    }
}

/// @notice Check if the allocation has ended
/// @dev Reverts if the allocation has not ended
function _checkOnlyAfterAllocation() internal view virtual {
    if (block.timestamp < allocationEndTime) revert ALLOCATION_NOT_ENDED();
}
```

Tool used

Manual Review

Recommendation

1. Don't have this 1-second overlap
2. Add a withdraw function to QVBaseStrategy

Discussion

jack-the-pug

@neeksec https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/656 is a dup of this.

jack-the-pug

Fixed: https://github.com/allo-protocol/allo-v2/pull/337

neeksec

@Evert0x

I agree with @jack-the-pug 's https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/261#issuecomment-1778751023. #656 should be a dup of this one.

Issue M-8: Problems with tokens that transfer less than amount. (Separate from fee-on-transfer issues!)

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/379

Found by

jkoppel Some tokens such as cUSDCv3 contain a special case for amount == type(uint256).max in their transfer functions that results in only the user's balance being transferred. This can be used to shut down several pool operations.

There are also problems with fee-on-transfer tokens, but that's a separate issue.

The contest FAQ states that all weird tokens should work with this protocol. I also asked the sponsor about this specific category of issues, and they said "this does like something which can be taken advantage of !"

Vulnerability Detail

Several things that can go wrong with this:

1. An attacker can put dust of this token in a wallet, and then call allo.fundPool() with type(uint256).max of this token. If the pool has not already been funded, then poolAmount will not be at type(uint256).max despite nothing being in the pool. It is now not possible to fund the pool.
2. Someone can do this in the DonationVotingMerkleDistributionVaultStrategy to set someones claim of this token to type(uint256).max . It is now impossible for anyone else to donate this token to them.

Impact

Pools cannot work with such tokens

Code Snippet

See, e.g.: https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/donation-voting-merkle-distribution-vault/DonationVotingMerkleDistributionVaultStrategy.sol#L125

Tool used

Manual Review

Recommendation

Explicitly do not support these tokens

Discussion

jkoppel

Escalate.

This is not a duplicate of #19. This involves a separate class of token. The available attacks are different, as discussed in the write-up. This is discussed in the issue write-up and confirmed with sponsor.

sherlock-admin2

> Escalate.
>
> This is not a duplicate of #19. This involves a separate class of token. The available attacks are different, as discussed in the write-up. This is discussed in the issue write-up and confirmed with sponsor.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

Agree with escalation, this is a valid issue on its on and not at all related to FOT tokens since all ERC-20 tokens are supported.

0x00ffDa

Valid separate issue, but possibly Low since the impacts listed are DoS / griefing rather than loss of funds.

jkoppel

It causes loss of the base fee.

0x00ffDa

> It causes loss of the base fee.

... which can be refunded by the protocol owner using `Allo.recoverFunds()`.

jkoppel

There's not clear guidance in the docs, but I can think of other cases where the existence of admin workarounds was insufficient to disqualify something as a medium. n.B. The fee is sent to the treasury, not the Allo contract. Can't use `recoverFunds`.

neeksec

Suggest to make this low/info.

Since this is a rare token type and the impact(losing base fee which could be refunded by Allo team) is low.

Evert0x

Planning to reject escalation and keep issue state as is.

The impact of this issue is not clear to me.

> Pools cannot work with such tokens

Does this mean other pool users can't get out? What is the impact on people?

In case the impact is significant I will consider assigning medium.

jkoppel

I mentioned two attacks. It means that multiple attacks are available if someone creates a pool with this token or tries to use it in donation voting, even though the contract is supposed to work with all token types.

MLON33

https://github.com/allo-protocol/allo-v2/pull/355
https://github.com/allo-protocol/allo-v2/pull/381

Evert0x

Copied from README

> Do you expect to use any of the following tokens with non-standard behaviour with the smart contracts? Yes as we support all ERC20 tokens.

It's clear that the issue is in scope. The question to answer is, what is the impact of this issue?

The impact is basically

- DOS on pool funding
- DOS on donations
- Small loss of funds that can technically be recovered but it isn't easy to do so.

Although the language in the judging guidelines should be improved, I plan on making this a separate Medium because of the following rule.

> Breaks core contract functionality, rendering the contract useless (should not be easily replaced without loss of funds) or leading to unknown potential exploits/loss of funds. Ex: Unable to remove malicious user/collateral from the contract.

The two DOS attacks can break the core contract functionality. In case a pool is popular, it's a blow to the protocol if that pool is suddenly unavailable. It's not easily replaced.

Evert0x

Result: Medium Unique

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- jkoppel: accepted

Evert0x

@jkoppel I might have misinterpreted the report and was under the assumption it would brick other deposits as well (e.g. USDC, WETH,..)

But I think the scope is only limited to tokens that have this special case "amount == type(uint256).max" in their transfer function

In that case the following rule should be applied as the issue is only related to these tokens.

> Non-Standard tokens: Issues related to tokens with non-standard behaviors, such as weird-tokens are not considered valid by default unless these tokens are explicitly mentioned in the README.

jkoppel

But it is mentioned in the README.

The protocol team was asked this question as part of the standard intake questionnaire.

> Do you expect to use any of the following tokens with non-standard behaviour with the smart contracts?

As part of this question, they were presented a list of tokens with weird behavior, including this one.

They responded:

> Yes as we support all ERC20 tokens.

This should be taken to mean the same as if they had listed every token they were asked about.

AhmadDecoded

@jkoppel Posted this in discord now posting it here.

hey i think there is need to check the validity of the claim:

```
Some tokens such as cUSDCv3 contain a special case for amount ==
↪  type(uint256).max in their transfer functions that results in only the
↪  user's balance being transferred. This can be used to shut down several pool
↪  operations.
```

Which have been copied from :

https://github.com/d-xo/weird-erc20

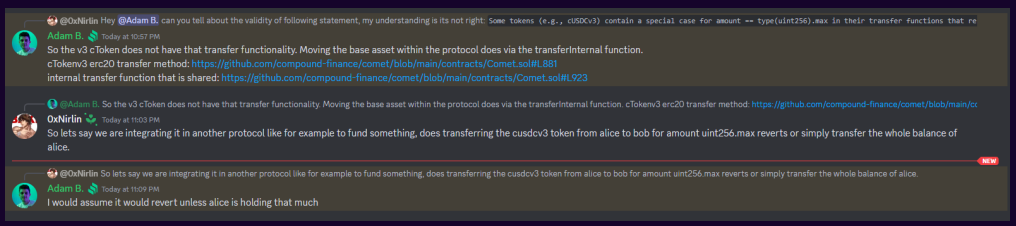
But as I confirmed from the compound dev, that is only true for internal transfer with in the compound v3 vaults

The code for the ctoken itself does not work like that and works as any standard erc20 token

So the token that you gave example of cusdcv3, that token itself would work fine with the protocol, so in this case the submission becomes arbitrary and invalid anyways, anyone can craft example of a token that may not work with one or other functionality.

Considering the sherlock rules just add another layer of argument.

AhmadDecoded



Here you can see what the dev have to say, also you can look into code yourself, the functionality you mentioned is for internal transfer with in compound v3 vaults in comet files, transfer between accounts works as any normal working erc20 tokens.

nevillehuang

I found the ERC20 implementation of `transferFrom` of the cUSDCv3 token:

```
function transferFrom(address src, address dst, uint amount) override external
↪  returns (bool) {
    transferInternal(msg.sender, src, dst, baseToken, amount);
    return true;
}
```

which calls the internal `transferInternal` function here so I think the dev might be incorrect here. If you call fundpool with type(uint256).max it seems like it does only transfer the amount of tokens the user owns.

```
function transferInternal(address operator, address src, address dst, address
↪  asset, uint amount) internal {
    if (isTransferPaused()) revert Paused();
    if (!hasPermission(src, operator)) revert Unauthorized();
    if (src == dst) revert NoSelfTransfer();

    if (asset == baseToken) {
        if (amount == type(uint256).max) {
            amount = balanceOf(src);
        }
        return transferBase(src, dst, amount);
    } else {
        return transferCollateral(src, dst, asset, safe128(amount));
    }
}
```

Now comes the dilemma between whether this behavior is "weird" token or not (which is not considered valid in sherlocks guidelines) or is the contest READ.ME on the question about non-standard tokens takes priority (sherlock guidelines states the contest details is the single source of truth). One thing important to note that while I could not find much data on the use of this token, it has a total market valuation of >, which is quite significant.

I will let @Evert0x decide on this one but I am leaning towards validating this issue. But perhaps moving forward, more defined rules revolving weird ERC20 tokens needs to be considered.

AhmadDecoded

@nevillehuang you are checking wrong contract Comet file represent the market for that token, not the token itself. Those all transfer functions are for internal transfers within that market. Should have asked before commenting.

As far as the question whether the behaviour is weird or not if it exists, this example have been taken directly from the weird tokens github repo.

AhmadDecoded

@quentin-abei you are trying to create mess to impose your decision on judge here. We are discussing validity as the token implementation is being misjudged here. If you want to comvince judge for your issue do it under your own issue, don't try negative reinforcement.

nevillehuang

> @nevillehuang you are checking wrong contract Comet file represent the market for that token, not the token itself. Those all transfer functions are for internal transfers within that market. Should have asked

before commenting.

Oh, could you then point me to the correct contract and function details? My understanding is cUSDv3 is a proxy of the implementation contract i linked. Perhaps @jkoppel should assist in this since burden of proof is on the watson.

jkoppel

Not having seen the discussion here, I just spent about 30 minutes hunting for the cUSDCv3 contract. The final answer would require unraveling their deployment scripts, but I'm pretty sure Neville is correct, sans a nitpick.

The description for the Comet repo is "An efficient money market protocol for Ethereum and compatible chains (aka Compound III, Compound v3)." So this looks like the right place.

There are no references to the string "cUSDCv3" in the code. But there are several in configuration files.

Comet.sol implements 5 of the 7 ERC20 methods, all but `name` and `symbol`.

This diagram states that Comet delegatecalls to CometExt: https://github.com/compound-finance/comet/blob/22cf923b6263177555272dde8b0791703895517d/diagrams/inheritance_diagram.uml . You can see that Comet does indeed implement fallback() and delegatecall's all other methods to another contract: https://github.com/compound-finance/comet/blob/22cf923b6263177555272dde8b0791703895517d/contracts/Comet.sol#L1318

CometExt provides the missing name() and symbol() methods, which it pulls from data: https://github.com/compound-finance/comet/blob/22cf923b6263177555272dde8b0791703895517d/contracts/CometExt.sol#L71

So it looks indeed that Comet.sol is in fact the implementation code for cUSDCv3, and it does have the issue reported in the Weird ERC20 repo.

In conclusion: cUSDCv3 is almost certainly a deployment of Comet.sol where some functionality is provided by delegating to a deployment of CometExt.sol.

I have also E-mailed the person who added this issue to the Weird ERC20 repo to confirm.

I think what probably caused the confusion is that the Compound dev saw the function name `transferInternal` and misunderstood it. It appears this dev has edited the documentation in this repo, but has not actually touched protocol code. https://github.com/compound-finance/comet/commits?author=ajb413

Evert0x

It seems like @jkoppel is correct that the `cUSDCv3` token contains the behavior described in his initial report.

It's a hard judgment to make as the context QA doesn't align with the judging guidelines

README states

> Do you expect to use any of the following tokens with non-standard behavior with the smart contracts? Yes as we support all ERC20 tokens.

Judging rules state

> Issues related to tokens with non-standard behaviors, such as weird-tokens are not considered valid by default unless these tokens are **explicitly** mentioned in the README.

`cUSDCv3` is not explicitly mentioned in the README. But the protocol indicated they plan on supporting any token with non-standard behavior.

The last resort is to depend on the opinion of the protocol, they confirmed the issue and are even planning on fixing the issue.

AhmadDecoded

Ok my mistake, I ran a simulation on tenderly, it does actually works like described.

I will leave it here, now upto judge for the final decision about the ruling on the weird tokens.

Issue M-9: `QVSimpleStrategy`: If someone fund a pool when the fund is partially/fully distributed, part of the fund may be locked

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/446

Found by

0xkaden, ast3ros, branch_indigo, jkoppel, lemonmon, lil.eth, nobody2018, rvierdiiev, wangxx2026 When `Allo::fundPool` is called when the funds are partially or fully distributed, the added funds may be locked.

Vulnerability Detail

`Allo::fundPool` can be called by anyone at anytime, and it will increase the `BaseStrategy.poolAmount` via `BaseStrategy::increasePoolAmount()`.

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/core/Allo.sol#L339-L345

The `BaseStrategy.poolAmount` storage variable is used to determine the payout of each recipient by `QVBaseStrategy::_getPayout`:

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-base/QVBaseStrategy.sol#L559-L574

When the fund is distributed by the pool manager via `QVBaseStrategy::_distribute`, the `paidOut` flag for the recipientId whose share was distributed will be set to be true.

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-base/QVBaseStrategy.sol#L458

The problem occurs when some funds are added when some funds are distributed. In the case, the funds will be partially or fully locked.

Impact

If some funds are added after the distribution is started, the added funds may be locked.

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/core/Allo.sol#L339-L345

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-base/QVBaseStrategy.sol#L559-L574

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-base/QVBaseStrategy.sol#L458

Tool used

Manual Review

Recommendation

Consider adding recoverFund function like other strategies. Alternatively, allow the `fundPool` function only before the distribution starts.

Discussion

jacksanford1

https://github.com/allo-protocol/allo-v2/pull/345

jack-the-pug

Fixed.

Issue M-10: `_distribute()` function in RFPSimpleStrategy contract has wrong requirement causing DOS

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/492

Found by

0xarno, 0xdeadbeef, 0xkaden, BenRai, GimelSec, HHK, KingNFT, WATCHPUG, ZdravkoHr., ace13567, alexxander, dany.armstrong90, fishgang, grearlake, honeymewn, jkoppel, lemonmon, nobody2018, osmanozdemir1, p0wd3r, pengun, pontifex, qbs, tnquanghuy0512

Vulnerability Detail

The function `_distribute()`:

```
function _distribute(address[] memory, bytes memory, address _sender)
    internal
    virtual
    override
    onlyInactivePool
    onlyPoolManager(_sender)
{
    ...

    IAllo.Pool memory pool = allo.getPool(poolId);
    Milestone storage milestone = milestones[upcomingMilestone];
    Recipient memory recipient = _recipients[acceptedRecipientId];

    if (recipient.proposalBid > poolAmount) revert NOT_ENOUGH_FUNDS();

    uint256 amount = (recipient.proposalBid * milestone.amountPercentage) / 1e18;

    poolAmount -= amount;//<@@ NOTICE the poolAmount get decrease over time

    _transferAmount(pool.token, recipient.recipientAddress, amount);

    ...
}
```

Let's suppose this scenario:

- Pool manager funding the contract with 100 token, making `poolAmount` variable equal to 100
- Pool manager set 5 equal milestones with 20% each

- Selected recipient's proposal bid is 100, making `recipients[acceptedRecipientId].proposalBid` variable equal to 100
- After milestone 1 done, pool manager pays recipient using `distribute()`. Value of variables after: `poolAmount = 80` `,recipients[acceptedRecipientId].proposalBid = 100`
- After milestone 2 done, pool manager will get DOS trying to pay recipient using `distribute()` because of this line:

```
if (recipient.proposalBid > poolAmount) revert NOT_ENOUGH_FUNDS();
```

Impact

This behaviour will cause DOS when distributing the 2nd milestone or higher

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/6430c8004017e96ae2f5aac365bdefd0b6eeea72/allo-v2/contracts/strategies/rfp-simple/RFPSimpleStrategy.sol#L417C1-L450C6

Tool used

Manual Review

Recommendation

```
-         if (recipient.proposalBid > poolAmount) revert NOT_ENOUGH_FUNDS();
+         if ((recipient.proposalBid * milestone.amountPercentage) / 1e18 >
 ↪  poolAmount) revert NOT_ENOUGH_FUNDS();
```

Discussion

jacksanford1

https://github.com/allo-protocol/allo-v2/pull/344

jack-the-pug

Fixed.

Issue M-11: Recipient cannot always claim expected donation amount from a vault

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/547

Found by

0x00ffDa, 0xdeadbeef, Arz, BenRai, Kow, Silvermist, ZdravkoHr., alymurtazamemon, hals, jkoppel, lemonmon, p0wd3r

Vulnerability Detail

First, why is this issue valid despite Sherlock rules excluding non-standard ERC20 behavior? While the Sherlock rules currently exclude "issues related to tokens with non-standard behaviors", the rules also state that "In case of conflict between information in the README, vs Sherlock rules, **the README overrides Sherlock rules**." And the contest README specifically says that all ERC20 with non-standard behavior are supported:

> "Q: Do you expect to use any of the following tokens with non-standard behaviour with the smart contracts? Yes as we support all ERC20 tokens."

DonationVotingMerkleDistributionVaultStrategy contracts store donation amounts in a claims mapping per recipient and per donated token. This is intended to also represent all the amounts claimable by registered recipients. However, in the case of donated ERC20 tokens with a non-zero transfer fee, a donated amount is not claimable in full since the transfer from the donor to the vault will reduce the amount that can be claimed by the recipient.

Impact

Recipients cannot claim the amount expected from a vault in a DonationVotingMerkleDistributionVaultStrategy pool when a donated token has a transfer fee.

Code Snippet

DonationVotingMerkleDistributionVaultStrategy._afterAllocate() function - the last line contains the faulty assumption.

```
/// @notice After allocation hook to store the allocated tokens in the vault
/// @param _data The encoded recipientId, amount and token
/// @param _sender The sender of the allocation
function _afterAllocate(bytes memory _data, address _sender) internal override {
```

```
    // Decode the '_data' to get the recipientId, amount and token
    (address recipientId, Permit2Data memory p2Data) = abi.decode(_data,
↪   (address, Permit2Data));

    // Get the token address
    address token = p2Data.permit.permitted.token;
    uint256 amount = p2Data.permit.permitted.amount;

    if (token == NATIVE) {
        if (msg.value < amount) {
            revert AMOUNT_MISMATCH();
        }
        SafeTransferLib.safeTransferETH(address(this), amount);
    } else {
        PERMIT2.permitTransferFrom(
            // The permit message.
            p2Data.permit,
            // The transfer recipient and amount.
            ISignatureTransfer.SignatureTransferDetails({to: address(this),
↪   requestedAmount: amount}),
            // Owner of the tokens and signer of the message.
            _sender,
            // The packed signature that was the result of signing
            // the EIP712 hash of `_permit`.
            p2Data.signature
        );
    }

    // Update the total payout amount for the claim
    claims[recipientId][token] += amount;
}
```

Tool used

Manual Review

Recommendation

In the `_afterAllocate()` cited above, compare vault's balance of the token being donated before and after calling `permitTransferFrom()`. If it is lower than the donated `amount`, assume this difference is a transfer fee and reduce the claimable amount by the fee.

Discussion

0x00ffDa

Escalate for 10 USDC

This is not a duplicate of https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/19 and the fix for that issue does not affect this issue.

This fee-on-transfer issue is specific to transfers from donors to recipients via `DonationVotingMerkleDistributionVaultStrategy`.

sherlock-admin2

> Escalate for 10 USDC
>
> This is not a duplicate of https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/19 and the fix for that issue does not affect this issue.
>
> This fee-on-transfer issue is specific to transfers from donors to recipients via `DonationVotingMerkleDistributionVaultStrategy`.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

neeksec

Agree with Escalation that this is not a duplicate of #19 and should be a seperate issue.

Duplicates of this one, #124, #208, #268, #295, #342, #379, #422, #449, #482, #724, #903, #913

MLON33

https://github.com/allo-protocol/allo-v2/pull/381

Evert0x

Planning to accept escalation and create a new medium family (https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/547 https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/124, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/208, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/268, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/295, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/342, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/379, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/422, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/449, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/482, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/724, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/903, https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/913)

Evert0x

Result: Medium Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0x00ffDa: accepted

jack-the-pug

Fixed.

Issue M-12: `QVBaseStrategy::reviewRecipients()` doesn't check if the recipient is already accepted or rejected, and overwrites the current status

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/699

Found by

0x00ffDa, BenRai, gkrastenov, osmanozdemir1, xAriextz

Vulnerability Detail

In the QV strategy contracts, recipients register themselves and wait for a pool manager to accept the registration. Pool managers can accept or reject recipients with the `reviewRecipients()` function. There is also a threshold (`reviewThreshold`) for recipients to be accepted. For example, if the `reviewThreshold` is 2, a pending recipient gets accepted when two managers accept this recipient and the `recipientStatus` is updated.

However, `QVBaseStrategy::reviewRecipients()` function doesn't check the recipient's current status. This one alone may not be an issue because managers may want to change the status of the recipient etc.
But on top of that, the function also doesn't take the previous review counts into account when updating the status, and overwrites the status immediately after reaching the threshold. I'll share a scenario later about this below.

Here is the `reviewRecipients()` function:
https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-base/QVBaseStrategy.sol#L254C1-L288C6

```
file: QVBaseStrategy.sol
    function reviewRecipients(address[] calldata _recipientIds, Status[]
↪   calldata _recipientStatuses)
        external
        virtual
        onlyPoolManager(msg.sender)
        onlyActiveRegistration
    {
        // make sure the arrays are the same length
        uint256 recipientLength = _recipientIds.length;
        if (recipientLength != _recipientStatuses.length) revert INVALID();

        for (uint256 i; i < recipientLength;) {
            Status recipientStatus = _recipientStatuses[i];
            address recipientId = _recipientIds[i];
```

```
            // if the status is none or appealed then revert
            if (recipientStatus == Status.None || recipientStatus ==
↪   Status.Appealed) { //@audit these are the input parameter statuse not the
↪   recipient's status.
                revert RECIPIENT_ERROR(recipientId);
            }

            reviewsByStatus[recipientId][recipientStatus]++;

-->         if (reviewsByStatus[recipientId][recipientStatus] >=
↪   reviewThreshold) { //@audit recipientStatus is updated right after the
↪   threshold is reached. It can overwrite if the status is already set.
                Recipient storage recipient = recipients[recipientId];
                recipient.recipientStatus = recipientStatus;

                emit RecipientStatusUpdated(recipientId, recipientStatus,
↪   address(0));
            }

            emit Reviewed(recipientId, recipientStatus, msg.sender);

            unchecked {
                ++i;
            }
        }
    }
```

As I mentioned above, the function updates the `recipientStatus` immediately after reaching the threshold. Here is a scenario of why this might be an issue.

Example Scenario
The pool has 5 managers and the `reviewThreshold` is 2.

1. The first manager rejects the recipient
2. The second manager accepts the recipient
3. The third manager rejects the recipient. -> `recipientStatus` updated -> `status = REJECTED`
4. The fourth manager rejects the recipient -> status still `REJECTED`
5. The last manager accepts the recipient ->`recipientStatus` updated again -> `status = ACCEPTED`

3 managers rejected and 2 managers accepted the recipient but the recipient status is overwritten without checking the recipient's previous status and is ACCEPTED now.

Coded PoC

You can prove the scenario above with the PoC. You can use the protocol's own setup for this.
- Copy the snippet below and paste it into the `QVBaseStrategy.t.sol` test file.
- Run forge test `--match-test test_reviewRecipient_reviewTreshold_OverwriteTheLastOne`

```
//@audit More managers rejected but the recipient is accepted
    function test_reviewRecipient_reviewTreshold_OverwriteTheLastOne() public
↪   virtual {
        address recipientId = __register_recipient();

        // Create rejection status
        address[] memory recipientIds = new address[](1);
        recipientIds[0] = recipientId;
        IStrategy.Status[] memory Statuses = new IStrategy.Status[](1);
        Statuses[0] = IStrategy.Status.Rejected;

        // Reject three times with different managers
        vm.startPrank(pool_manager1());
        qvStrategy().reviewRecipients(recipientIds, Statuses);

        vm.startPrank(pool_manager2());
        qvStrategy().reviewRecipients(recipientIds, Statuses);

        vm.startPrank(pool_manager3());
        qvStrategy().reviewRecipients(recipientIds, Statuses);

        // Three managers rejected. Status will be rejected.
        assertEq(uint8(qvStrategy().getRecipientStatus(recipientId)),
↪   uint8(IStrategy.Status.Rejected));
        assertEq(qvStrategy().reviewsByStatus(recipientId,
↪   IStrategy.Status.Rejected), 3);

        // Accept two times after three rejections
        Statuses[0] = IStrategy.Status.Accepted;
        vm.startPrank(pool_admin());
        qvStrategy().reviewRecipients(recipientIds, Statuses);

        vm.startPrank(pool_manager4());
        qvStrategy().reviewRecipients(recipientIds, Statuses);

        // 3 Rejected, 2 Accepted, but status is Accepted because it overwrites
↪   right after passing threshold.
        assertEq(uint8(qvStrategy().getRecipientStatus(recipientId)),
↪   uint8(IStrategy.Status.Accepted));
```

```
        assertEq(qvStrategy().reviewsByStatus(recipientId,
↪   IStrategy.Status.Rejected), 3);
        assertEq(qvStrategy().reviewsByStatus(recipientId,
↪   IStrategy.Status.Accepted), 2);
    }
```

You can find the test results below:

```
Running 1 test for
↪   test/foundry/strategies/QVSimpleStrategy.t.sol:QVSimpleStrategyTest
[PASS] test_reviewRecipient_reviewTreshold_OverwriteTheLastOne() (gas: 249604)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 10.92ms
```

Impact

Recipient status might be overwritten with less review counts.

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/strategies/qv-base/QVBaseStrategy.sol#L254C1-L288C6

```
file: QVBaseStrategy.sol
    function reviewRecipients(address[] calldata _recipientIds, Status[]
↪   calldata _recipientStatuses)
        external
        virtual
        onlyPoolManager(msg.sender)
        onlyActiveRegistration
    {
        // make sure the arrays are the same length
        uint256 recipientLength = _recipientIds.length;
        if (recipientLength != _recipientStatuses.length) revert INVALID();

        for (uint256 i; i < recipientLength;) {
            Status recipientStatus = _recipientStatuses[i];
            address recipientId = _recipientIds[i];

            // if the status is none or appealed then revert
            if (recipientStatus == Status.None || recipientStatus ==
↪   Status.Appealed) { //@audit these are the input parameter statuse not the
↪   recipient's status.
                revert RECIPIENT_ERROR(recipientId);
            }

            reviewsByStatus[recipientId][recipientStatus]++;
```

```
-->         if (reviewsByStatus[recipientId][recipientStatus] >=
↪  reviewThreshold) { //@audit recipientStatus is updated right after the
↪  threshold is reached. It can overwrite if the status is already set.
                Recipient storage recipient = recipients[recipientId];
                recipient.recipientStatus = recipientStatus;

                emit RecipientStatusUpdated(recipientId, recipientStatus,
↪  address(0));
            }

            emit Reviewed(recipientId, recipientStatus, msg.sender);

            unchecked {
                ++i;
            }
        }
    }
```

Tool used

Manual Review

Recommendation

Checking the review counts before updating the state might be helpful to mitigate this issue

Discussion

0xKurt

@nfrgosselin How to deal with this depends of how the pool admin wants to approve or reject applications:

1. Count each status --> 4 accepted and 2 rejected with threshold at 8 this would be approved
2. Whatever the last status set is --> the last manager to set the status
3. Once the status is set it cannot be changed

These are a couple example we thought of. WDYT?

sherlock-admin2

> Escalate

> #589 and #699 are essentially describing the same root cause. All relevant dupes associated with this two issues should be combined, as both are describing the possibility of pool manager reviewing a recipient multiple times leading to overwritten recipient status.

```
You've deleted an escalation for this issue.
```

osmanozdemir1

> Escalate
>
> #589 and #699 are essentially describing the same root cause. All relevant dupes associated with this two issues should be combined, as both are describing the possibility of pool manager reviewing a recipient multiple times leading to overwritten recipient status.

This issue doesn't describe same manager reviewing a recipient multiple times. As you can see in the coded PoC above, every review is made by different managers. This issue is far different than the #589 and describes recipient can be changed even with less review counts.

nevillehuang

I still stand that both are duplicates. Both are describing the same root cause of not checking the previous status and not respecting review threshold, allowing overwriting of recipient status

BenRai1

I would side with @osmanozdemir1 on that. The issues https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/589 and https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/699 are different since they describe two different issues where the fix is different.

The first issue would be fixed by tracking what manager already voted for what recipient. This would not fix the issue where the last manager who reviews a recipient can turn the result even if more managers voted for an other result. Just becasue both issues are in the same function this does not mean that they are the same

nevillehuang

> I would side with @osmanozdemir1 on that. The issues #589 and #699 are different since they describe two different issues where the fix is different.
>
> The first issue would be fixed by tracking what manager already voted for what recipient. This would not fix the issue where the last manager who reviews a recipient can turn the result even if more managers voted

> for an other result. Just becasue both issues are in the same function this does not mean that they are the same

On second thought, you are right, the issues are not duplicates. One is talking about pool managers potentially casting many votes influencing recipient status by bypassing `reviewThreshold`, while the other is pool managers casting votes as expected but doesn't respect the majority outcome. Removing escalation

0xf1b0

In my report #315, I mentioned both cases described in #589 and this one. If it's not a duplicate I should be added to the #699 "found by" list.

nevillehuang

Escalate

On third thought, report #315 mentions both issues, and again convinces me that #589 and #699 are stemming from the same root cause of ineffective reviewThresholds. Both issues stems from overriding recipient status, regardless it being the same pool manager or multiple different pool manager. #315 should be made a primary issue and all other issues should be dupped under it.

sherlock-admin2

> Escalate
>
> On third thought, report #315 mentions both issues, and again convinces me that #589 and #699 are stemming from the same root cause of ineffective reviewThresholds. Both issues stems from overriding recipient status, regardless it being the same pool manager or multiple different pool manager. #315 should be made a primary issue and all other issues should be dupped under it.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

osmanozdemir1

> Escalate
>
> On third thought, report #315 mentions both issues, and again convinces me that #589 and #699 are stemming from the same root cause of ineffective reviewThresholds. Both issues stems from overriding recipient status, regardless it being the same pool manager or multiple different pool manager. #315 should be made a primary issue and all other issues should be dupped under it.

I disagree with this comment and disagree with both of these issues being duplicates. The solution of the first issue is tracking which manager reviewed which recipient with a mapping. This solution is implemented in this fix. The fix of the first issue does not solve the second issue. The solution of the second issue can be implemented in different ways depending on the developer team's intentions. Which is mentioned here, and the fix is implemented here. As you can see the fixes of these two issues are quite different.

Just because one submission (#315) mentioning two different issues together doesn't mean those two issues are the same, and doesn't mean all of them should be duped together. It only means that the submitter preferred to submit them together to strengthen his/her submission. As you can see in the Sherlock documentation, best practices number 3: *"Do not submit multiple issues in a single submission. Even if the 2 completely different issues occur on the same line, please make sure to separately submit them"*

I'm not sure if the Sherlock allows labelling one submission with two different duplicate labels in these kind of situations. #315 might be duped under both of these two primary issues or it can be duped under the one with less duplicates according to the Sherlock rules and the judge's decision.

0xf1b0

By the way, my report #315 also includes a mention of and a proposed fix for #729 issue. Since it's my first time participating in a contest and I don't have the capability to escalate, I'll leave it here with the hope that it will be taken into consideration.

0xf1b0

I will certainly need to reconsider the issue submission process.

However, it's important to note that "best practices" should not be interpreted as rigid rules. The documentation does not explicitly state that deviating from these practices will result in no rewards or that only a single issue will be acknowledged.

Moreover, the question of whether these issues are truly separate remains a subject of debate.

neeksec

I suggest to keep the orignal judging.

> Both are describing the same root cause of not checking the previous status and not respecting review threshold

This one is not checking the previous status and #589 is not respecting review threshold. Although these two bugs both alow to manipulate review status, the root cause is different. The fixes are also different which was well described by https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/699#issuecomment-1763517995.

jacksanford1

https://github.com/allo-protocol/allo-v2/pull/349

Evert0x

Planning to reject escalation and keep issue state.

Root cause is different in the mentioned issues.

Evert0x

Result: Medium Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- nevillehuang: rejected

jack-the-pug

Not Fixed.

Managers who haven't voted yet can still change a recipient's status by voting for another status.

Issue M-13: `QVBaseStrategy` contract : recipient `reviewStatus` is not reset upon re-registration

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/729

Found by

ArmedGoose, WATCHPUG, ast3ros, hals, honeymewn `QVBaseStrategy` contract : the reviewStatus of the recipient is not reset (set to zero) when he re-registres again.

Vulnerability Detail

- In `QVBaseStrategy` strategy contract: when the user first registers; his status is updated from `None` to `Pending`.
- Then the pool manager can review recipients (via `reviewRecipients` function) and update their statuses from `Pending` to `Accepted` and from `Accepted` to `Pending` only (not updating to `Rejected` or `Appealed`) if these recipients statuses got votes equal to `reviewThreshold`:

  QVBaseStrategy::reviewRecipients /L275-280

```
if (reviewsByStatus[recipientId][recipientStatus] >= reviewThreshold) {
    Recipient storage recipient = recipients[recipientId];
    recipient.recipientStatus = recipientStatus;

    emit RecipientStatusUpdated(recipientId, recipientStatus, address(0));
}
```

- The strategy contract allows registered recipients to re-register again with new terms; and when doing so, their statuses are updated from `Accepted ==> Pending` or from `Rejected` to `Appealed`:

  QVBaseStrategy::_registerRecipient /L275-280

```
if (currentStatus == Status.None) {
          // recipient registering new application
          recipient.recipientStatus = Status.Pending;
          emit Registered(recipientId, _data, _sender);
      } else {
          if (currentStatus == Status.Accepted) {
              // recipient updating accepted application
              recipient.recipientStatus = Status.Pending;
          } else if (currentStatus == Status.Rejected) {
              // recipient updating rejected application
              recipient.recipientStatus = Status.Appealed;
          }
```

```
            // emit the new status with the '_data' that was passed in
            emit UpdatedRegistration(recipientId, _data, _sender,
↪   recipient.recipientStatus);
        }
```

Impact

But as can be noticed; the reviewRecipient status of the re-registered recipient is not reset which will result in this re-registered recipient getting `Accepted` status on their new registration in the next review round/rounds with lesser votes to reach `reviewThreshold`.

Code Snippet

QVBaseStrategy::_registerRecipient /L275-280

```
if (currentStatus == Status.None) {
            // recipient registering new application
            recipient.recipientStatus = Status.Pending;
            emit Registered(recipientId, _data, _sender);
        } else {
            if (currentStatus == Status.Accepted) {
                // recipient updating accepted application
                recipient.recipientStatus = Status.Pending;
            } else if (currentStatus == Status.Rejected) {
                // recipient updating rejected application
                recipient.recipientStatus = Status.Appealed;
            }

            // emit the new status with the '_data' that was passed in
            emit UpdatedRegistration(recipientId, _data, _sender,
↪   recipient.recipientStatus);
        }
```

QVBaseStrategy::reviewRecipients

```
function reviewRecipients(address[] calldata _recipientIds, Status[] calldata
↪   _recipientStatuses)
    external
    virtual
    onlyPoolManager(msg.sender)
    onlyActiveRegistration
{
    // make sure the arrays are the same length
    uint256 recipientLength = _recipientIds.length;
```

```
    if (recipientLength != _recipientStatuses.length) revert INVALID();

    for (uint256 i; i < recipientLength;) {
        Status recipientStatus = _recipientStatuses[i];
        address recipientId = _recipientIds[i];

        // if the status is none or appealed then revert
        if (recipientStatus == Status.None || recipientStatus ==
↪   Status.Appealed) {
            revert RECIPIENT_ERROR(recipientId);
        }

        reviewsByStatus[recipientId][recipientStatus]++;

        if (reviewsByStatus[recipientId][recipientStatus] >= reviewThreshold) {
            Recipient storage recipient = recipients[recipientId];
            recipient.recipientStatus = recipientStatus;

            emit RecipientStatusUpdated(recipientId, recipientStatus,
↪   address(0));
        }

        emit Reviewed(recipientId, recipientStatus, msg.sender);

        unchecked {
            ++i;
        }
    }
}
```

Tool used

Manual Review

Recommendation

Update `_registerRecipient` function to reset `reviewsByStatus[recipientId][recipientStatus]` when the recipient re-registers:

```
if (currentStatus == Status.None) {
          // recipient registering new application
          recipient.recipientStatus = Status.Pending;
          emit Registered(recipientId, _data, _sender);
      } else {
          if (currentStatus == Status.Accepted) {
              // recipient updating accepted application
              recipient.recipientStatus = Status.Pending;
```

```
+             reviewsByStatus[recipientId][Status.Accepted]=0;
          } else if (currentStatus == Status.Rejected) {
              // recipient updating rejected application
              recipient.recipientStatus = Status.Appealed;
          }

          // emit the new status with the '_data' that was passed in
          emit UpdatedRegistration(recipientId, _data, _sender,
 ↪  recipient.recipientStatus);
      }
```

Discussion

codenamejason

A related issue -> https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/699

jacksanford1

https://github.com/allo-protocol/allo-v2/pull/349

jack-the-pug

Not fixed. See also the recommendation of https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/680

Issue M-14: [M-01] Allocation can start before registration ends. Which can break things in strategies.

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/751

Found by

0xkaden, Kral01, WATCHPUG, ace13567, honeymewn, lemonmon, tnquanghuy0512
Allocation can start before registration ends. Which can break things in strategies. When using `QVBaseStrategy` and `DonationVotingMerkleDistributionBaseStrategy` .

Vulnerability Detail

The way that timestamps are set or specifically, validated right now allows `Allocation` to start before `Registration` ends. I havent git a clear answer from the protocol team regarding whether this is intended or not, but even if it is intended, this can potentially break some stuff in `QVBaseStrategy` and `DonationVotingMerkleDistributionBaseStrategy`.

If `_registerRecipient` is called while `Allocation` is active on both QVBaseStrategy and DonationVotingMerkleDistributionBaseStrategy This can completely modify the effects of `reviewRecipients` and rewrite the status of already 'reviewed' recipients. This can cause rewards being falsely distributed or even stuck as well.

Of course there is a check to see if the caller is a member of the pool. Since the members are 'trusted' then the likelihood is low, thus the impact will be medium, if those roles are not to be trusted, then impact can potentially be high.

Impact

Recipient statuses can be modified and altered causing funds to sent falsely or get stuck.

Code Snippet

This is the check to validate timestamps in both strategies. As we can see, it allows `allocationStartTime` to be less than `registrationEndTime`.

```
if (
          block.timestamp > _registrationStartTime || _registrationStartTime >
↪  _registrationEndTime
              || _registrationStartTime > _allocationStartTime ||
↪  _allocationStartTime > _allocationEndTime
              || _registrationEndTime > _allocationEndTime
```

```
        ) {//@audit I think we can start allocation before registration ends
            revert INVALID();
```

Tool used

Manual Review

Recommendation

Consider adding check to revert if allocation can start before registration ends:

```
if (
          block.timestamp > _registrationStartTime || _registrationStartTime >
↪  _registrationEndTime
              || _registrationStartTime > _allocationStartTime ||
↪  _allocationStartTime > _allocationEndTime
              || _registrationEndTime > _allocationEndTime  ||
↪  _registrationEndTime > _allocationStartTime
      ) {
          revert INVALID();
```

Discussion

thelostone-mc

It is intended to have the ability to register even after allocation has started. But the funds getting stuck for a recipient who has reapplied can be tricky. We could explore a fix this to ensure reapplications are not possible once the allocation has started

0xKurt

> But the funds getting stuck for a recipient who has reapplied can be tricky.

We don't see a way how funds could get stuck in the contract.

thelostone-mc

Yup this doesn't seem to be an issue. This can be ignored as it works as intended

neeksec

Agree with sponsor.

kadenzipfel

Escalate This should be a valid medium. Locked funds can happen as follows:

- Recipient registers and is accepted

- Recipient receives allocation (possible because crossover of registration and allocation periods)
- Recipient re-registers (perhaps near the end of registration period)
  - Recipient now has status of pending until a new review is processed
  - Recipient still has allocated credits
- Registration period completes before recipient is re-reviewed (possible since `registerRecipient` and `reviewRecipients` both are `onlyActiveRegistration`, so there may not be time for the recipient to be re-reviewed
- Distribution can no longer be processed to re-registered recipient as they are no longer accepted revert
- No other way to withdraw allocated share of `poolAmount` and thus the funds will be permanently locked in the contract

I've included a similar explanation in my submission which may better explain the finding: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/361

sherlock-admin2

> Escalate This should be a valid medium. Locked funds can happen as follows:
>
> - Recipient registers and is accepted
> - Recipient receives allocation (possible because crossover of registration and allocation periods)
> - Recipient re-registers (perhaps near the end of registration period)
>   - Recipient now has status of pending until a new review is processed
>   - Recipient still has allocated credits
> - Registration period completes before recipient is re-reviewed (possible since `registerRecipient` and `reviewRecipients` both are `onlyActiveRegistration`, so there may not be time for the recipient to be re-reviewed
> - Distribution can no longer be processed to re-registered recipient as they are no longer accepted revert
> - No other way to withdraw allocated share of `poolAmount` and thus the funds will be permanently locked in the contract
>
> I've included a similar explanation in my submission which may better explain the finding: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/361

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

neeksec

Suggest to keep invalid.

Sponsor said that "It is intended to have the ability to register even after allocation has started."

Accepted recipient re-register should be re-reviewed. They should know this risk.

If funds are not successfully distributed, pool manager should be able to withdraw them. Fixing #446 solves the locking problem.

kadenzipfel

> Suggest to keep invalid.
>
> Sponsor said that "It is intended to have the ability to register even after allocation has started."
>
> Accepted recipient re-register should be re-reviewed. They should know this risk.
>
> If funds are not successfully distributed, pool manager should be able to withdraw them. Fixing #446 solves the locking problem.

Disagree. While functionality may be inline with sponsor intent, the result of funds being locked under normal usage is clearly not intended. Recipients being aware of the risk of losing *their own allocation* is valid, but they shouldn't be aware of and responsible for this leading to funds being permanently locked.

The fact that the solution to https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/446 happens to solve this should not invalidate the finding. If that were reasonable, then it would be equally reasonable to invalidate https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/446 if the recommended mitigation to this issue solved https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/446.

Note that this is issue was initially sponsor disputed because it was unclear how it would lead to funds being locked, which has since been clarified https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/751#issuecomment-1763188537 and was already made clear in other issues, including https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/361

Evert0x

@neeksec any follow up comment?

neeksec

Agree with @kadenzipfel on

> the result of funds being locked under normal usage is clearly not intended

Agree with escalation that this should be medium.

Evert0x

Thanks. Planning to accept escalation and make medium

Evert0x

Result: Medium Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- kadenzipfel: accepted

Issue M-15: CREATE3 is not available in the zkSync Era.

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/862

Found by

0xc0ffEE, 0xnirlin, carrotsmuggler, imsrybr0, jkoppel, pengun

Vulnerability Detail

According to the contest README, the project can be deployed in zkSync Era. (https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/README.md?plain=1#L11)

The zkSync Era docs explain how it differs from Ethereum.

The description of CREATE and CREATE2 (https://era.zksync.io/docs/reference/architecture/differences-with-ethereum.html#create-create2) states that Create cannot be used for arbitrary code unknown to the compiler.

POC:

```
// SPDX-License-Identifier: Unlicensed
pragma solidity ^0.8.0;

import "./MiniContract.sol";
import "./CREATE3.sol";

contract DeployTest {
    address public deployedAddress;
    event Deployed(address);

    function generateContract() public returns(address, address) {
        bytes32 salt = keccak256("SALT");

        address preCalculatedAddress = CREATE3.getDeployed(salt);

        // check if the contract has already been deployed by checking code size
↪   of address
        bytes memory creationCode =
↪   abi.encodePacked(type(MiniContract).creationCode, abi.encode(777));

        // Use CREATE3 to deploy the anchor contract
        address deployed = CREATE3.deploy(salt, creationCode, 0);
        return (preCalculatedAddress, deployed);
    }
}
```

You can check sample POC code at zkSync Era Testnet(https://goerli.explorer.zksync.io/address/0x0f670f8AfcB09f4BC509Cb59D6e7CEC1A52BFA51#contract)

Also, the logic to compute the address of Create2 is different from Ethereum, as shown below, so the CREATE3 library cannot be used as it is.

This cause registry returns an incorrect `preCalculatedAddress`, causing the anchor to be registered to an address that is not the actual deployed address.

```
address  keccak256(
    keccak256("zksyncCreate2")
↪   0x2020dba91b30cc0006188af794c2fb30dd8520db7e2c088b7fc7c103c00ca494,
    sender,
    salt,
    keccak256(bytecode),
    keccak256(constructorInput)
 )
```

Impact

`generateAnchor` doesn't work, so user can't do anything related to anchor.

Code Snippet

https://github.com/allo-protocol/allo-v2/blob/851571c27df5c16f6586ece2a1cb6fd0acf04ec9/contracts/core/Registry.sol#L350
https://github.com/allo-protocol/allo-v2/blob/851571c27df5c16f6586ece2a1cb6fd0acf04ec9/contracts/core/Registry.sol#L338

Tool used

Manual Review

Recommendation

This can be solved by implementing CREATE2 directly instead of CREATE3 and using `type(Anchor).creationCode`. Also, the compute address logic needs to be modified for zkSync.

Discussion

0xdeadbeef0x

Escalate

I argue that this and #411 show be low since there is no loss of funds. While this impacts the functionality of the protocol on zkSync, no funds are at risk.

Also, createPoolWithCustomStrategy can be used instead - see comment https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/411#issuecomment-1757462133

sherlock-admin2

> Escalate
>
> I argue that this and #411 show be low since there is no loss of funds. While this impacts the functionality of the protocol on zkSync, no funds are at risk.
>
> Also, createPoolWithCustomStrategy can be used instead - see comment https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/411#issuecomment-1757462133

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

neeksec

Agree with Escalation that this one and #411 is low.

Re-read the docs. DoS without fund loss is not counted as medium.

> Could Denial-of-Service (DOS), griefing, or locking of contracts count as a Medium (or High) issue? It would not count if the DOS, etc. lasts a known, finite amount of time <1 year. **If it will result in funds being inaccessible for >=1 year, then it would count as a loss of funds and be eligible for a Medium or High designation**.

Also checked a similar issue in Kyberswap. Can agree with the Escalation.

imsrybr0

Both of this and #411 break core contract functionality.

EDIT: Also, createPoolWithCustomStrategy cannot be used as it requires a valid profile which cannot be created because of this issue

jacksanford1

https://github.com/allo-protocol/allo-v2/pull/387

Evert0x

If I understand correctly the complete zksync era deployment will be useless because of this.

https://docs.sherlock.xyz/audits/judging/judging#v.-how-to-identify-a-medium-issue

> Breaks core contract functionality, rendering the contract useless

I believe this creates a very strong argument for Medium severity.

> Also checked https://github.com/sherlock-audit/2023-07-kyber-swap-judging/issues/26#issuecomment-1711944172.

KyberSwap didn't explicitly tag `zksync era` as an EVM chain they were planning to deploy on.

Planning to reject escalation and keep medium.

0xdeadbeef0x

@Evert0x

I would appreciate to not reject the escalation because up until now only issues that result in loss of funds would get accepted by Sherlock (hence my reasoning for escalating this)

It was based on previous contests and for the clear definition in Sherlock docs: https://docs.sherlock.xyz/audits/judging/judging#ii.-criteria-for-issue-severity

```
There is a viable scenario (even if unlikely) that could cause the protocol to enter a state where a material amount of funds can be lost.
```

Even in the DOS rule it mentions there has to be loss of funds

0xRizwan

> KyberSwap didn't explicitly tag `zksync era` as an EVM chain they were planning to deploy on.

@Evert0x That kyberswap report was submitted by me. It is supposed to deploy and will be deployed on `zkSync Era`, I would just draft a detail response on kyberswap issue. To be noted, I am in discussion on the kyber swap issue with `Hrishi` and `Jack` from last 2 weeks.

jack-the-pug

https://github.com/allo-protocol/allo-v2/pull/387/files#diff-342c707b787edfadc9b07ed13856c8915c85b47c3bb10e7190b005f6fd177e50R348-R350

@0xKurt I believe that the way the address is computed in the updated version still returns an incorrect preCalculatedAddress on zkSync Era.

0xKurt

we will look into it @jack-the-pug. If you have any suggestions for us, feel free to drop them. cc @codenamejason @thelostone-mc

Evert0x

Result: Medium Has Duplicates

> Even in the DOS rule it mentions there has to be loss of funds This not about the interpretation of the DOS rule but about the `How to identify a medium issue` rule.

Besides the reasons mentioned earlier. The following is also a key rule in the judging

> Hierarchy of truth: Contest README > Sherlock rules for valid issues > Historical decisions.

Although the correct language is missing the intention of this sentence is that the protocol can thrive in the context defined by the protocol team. We will update to language to make this clear for future contests.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xdeadbeef0x: rejected

Issue M-16: Anchor contract is unable to receive NFTs of any kind

Source: https://github.com/sherlock-audit/2023-09-Gitcoin-judging/issues/883

Found by

0xnirlin, sashik_eth

Vulnerability Detail

Anchor.sol essentially works like a wallet, and also attached to profile to give it extra credibility and profile owner more functionality.

As intended this contract will receive nfts, from different strategies and protocols. However, as it is currently implemented thee contracts will not be able to receive NFTs sent with safeTransferFrom(), because they do not implement the necessary functions to safely receive these tokens..

While in many cases such a situation would be Medium severity, looking at these wallets will be used could lead to more serious consequences. For example, having an anchor that is entitled to high value NFTs but is not able to receive them is clearly a loss of funds risk, and a High severity issue.

implement the onERC721Received() and onERC1155Received() functions in following code:

```
// SPDX-License-Identifier: AGPL-3.0-only
pragma solidity 0.8.19;

// Core Contracts
import {Registry} from "./Registry.sol";

//                    allo.gitcoin.co

/// @title Anchor contract
/// @author @thelostone-mc <aditya@gitcoin.co>, @0xKurt <kurt@gitcoin.co>,
↪  @codenamejason <jason@gitcoin.co>, @0xZakk <zakk@gitcoin.co>, @nfrgosselin
↪  <nate@gitcoin.co>
/// @notice Anchors are associated with profiles and are accessible exclusively
↪  by the profile owner. This contract ensures secure
///         and authorized interaction with external addresses, enhancing the
↪  capabilities of profiles and enabling controlled
///         execution of operations. The contract leverages the `Registry`
↪  contract for ownership verification and access control.
contract Anchor {
    /// ==========================
```

```
    /// === Storage Variables ====
    /// ==========================

    /// @notice The registry contract on any given network/chain
    Registry public immutable registry;

    /// @notice The profileId of the allowed profile to execute calls
    bytes32 public immutable profileId;

    /// ==========================
    /// ======== Errors ==========
    /// ==========================

    /// @notice Throws when the caller is not the owner of the profile
    error UNAUTHORIZED();

    /// @notice Throws when the call to the target address fails
    error CALL_FAILED();

    /// ==========================
    /// ======= Constructor ======
    /// ==========================

    /// @notice Constructor
    /// @dev We create an instance of the 'Registry' contract using the
↪   'msg.sender' and set the profileId.
    /// @param _profileId The ID of the allowed profile to execute calls
    constructor(bytes32 _profileId) {
        registry = Registry(msg.sender);
        profileId = _profileId;
    }

    /// ==========================
    /// ======== External ========
    /// ==========================

    /// @notice Execute a call to a target address
    /// @dev 'msg.sender' must be profile owner
    /// @param _target The target address to call
    /// @param _value The amount of native token to send
    /// @param _data The data to send to the target address
    /// @return Data returned from the target address
    function execute(address _target, uint256 _value, bytes memory _data)
↪   external returns (bytes memory) {
        // Check if the caller is the owner of the profile and revert if not
        if (!registry.isOwnerOfProfile(profileId, msg.sender)) revert
↪   UNAUTHORIZED();
```

```
        // Check if the target address is the zero address and revert if it is
        if (_target == address(0)) revert CALL_FAILED();

        // Call the target address and return the data
        (bool success, bytes memory data) = _target.call{value: _value}(_data);

        // Check if the call was successful and revert if not
        if (!success) revert CALL_FAILED();

        return data;
    }

    /// @notice This contract should be able to receive native token
    receive() external payable {}
}
```

Impact

Any time an ERC721 or ERC1155 is attempted to be transferred with safeTransferFrom() or minted with safeMint(), the call will fail.

Code Snippet

https://github.com/sherlock-audit/2023-09-Gitcoin/blob/main/allo-v2/contracts/core/Anchor.sol#L1C1-L88C2

Tool used

Manual Review

Recommendation

implement the onERC721Received() and onERC1155Received() functions

Discussion

sherlock-admin

1 comment(s) were left on this issue during the judging contest.

n33k commented:

> invalid, README says non ERC721 tokens will interact with the smart contracts

AhmadDecoded

Escalate

This is a valid high, yes the readme mentions that no erc721 will interact with the contract.

But as described many times by sponsor, the anchor should be able to receive nfts, that also adds to the reputation of the anchor and in general anchor are meant to receive nfts, both erc721 and erc1155.

sherlock-admin2

> Escalate
>
> This is a valid high, yes the readme mentions that no erc721 will interact with the contract.
>
> But as described many times by sponsor, the anchor should be able to receive nfts, that also adds to the reputation of the anchor and in general anchor are meant to receive nfts, both erc721 and erc1155.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

This ones a tough one. The new hierachy of truth based on sherlock docs states that:

> Hierarchy of truth: Contest README > Sherlock rules for valid issues > Historical decisions. While considering the validity of an issue in case of any conflict the sources of truth are prioritized in the above order. For example: In case of conflict between Sherlock rules vs Sherlock's historical decision, Sherlock criteria for issues must be considered the source of truth. In case of conflict between information in the README vs Sherlock rules, the README overrides Sherlock rules.

But it should be noted that the gitcoin docs is part of the README too, and under the anchor contract information, explicitly states here that:

> The anchor can accrue reputation for the project. It can represent the project in any attestations, be used as a soul-bound token, or used as a pointer in a registry

So i am inclined to think this is a valid issue. Sidenote: #731 is a duplicate of this issue

AhmadDecoded

Also the argument by sashik that read me did not mention about erc1155 is valid too.

AhmadDecoded

Hey @neeksec, appreciate your judging effort.

It should be valid high based on following arguments:

1. Registery for an achor is set on deployment, so specific anchor code js attached to the registery forever.
2. Anchor contract is not upgradable, so it can not be changed later and add the required receiving functions.
3. Historically the inability to receive nfts have been given the high status by sherlock. See splits contest example given in #731
4. Sherlock rules does not consider if something can be upgraded later on, audit vulnerability is given status based on current implementation unlike immunefi.

So it clearly breaks the core functionality and goes against the docs and team have already confirmed and implemented the fix, so should be given high status. It is one of the functionality that alloe will be highly reliant on.

neeksec

I deleted my initial response.

I re-read the conflicting docs.

> The anchor can accrue reputation for the project. It can represent the project in any attestations, be used as a soul-bound token, or used as a pointer in a registry

Since the NFTs are representing reputations or `represent the project in any attestations`, losing them dosen't sound like `a material lost of funds`.

AhmadDecoded

And also it makes the concept of anchor useless too than, breaking the core functionality.

Also strategies are highly cutomizible, so nfts can have the material value, sponsor confirmed this. So as more public strategies will be added this inability to receive nfts can also lead to loss of funds and definitely loss of opportunity.

High sounds more reasonable.

jacksanford1

https://github.com/allo-protocol/allo-v2/pull/378

Evert0x

Other protocols can just use `transferFrom` instead of `safeTransferFrom`, correct?

nevillehuang

> Other protocols can just use `transferFrom` instead of `safeTransferFrom`, correct?

Correction, you are right users can simply call `transferFrom()` for ERC721. But for ERC1155, only `safeTransferFrom()` options exists. So this DoS only applies to ERC1155 tokens

See here: https://eips.ethereum.org/EIPS/eip-1155#erc-1155-token-receiver

Evert0x

Planning to make medium as this issue only exists for ERC1155. With #731 as a duplicate

sashik-eth

> Planning to make medium as this issue only exists for ERC1155. With #731 as a duplicate

I'm afraid I have to disagree that applicability to only ERC1155 decreases the severity of this issue. The ERC1155 token is not necessarily less valuable than the ERC721 token. Moreover, the popularity of ERC1155 contracts is constantly increasing during the last two years as we can see on the chart from this page: https://dune.com/ilemi/erc-and-eip-starter-kit Also, it is worth mentioning that the external sender could be a protocol that has utilized only the `safeTransferFrom` function on the ERC721 contract, in this case, the problem with the ERC721 token would also occur.

Evert0x

> Also, it is worth mentioning that the external sender could be a protocol that has utilized only the safeTransferFrom function on the ERC721 contract, in this case, the problem with the ERC721 token would also occur.

This is not a valid reason to validate issues, also `safeTransferFrom` is not necessarily better to use in comparison to `transferFrom`.

> Issues that result out of a future integration/implementation that was not intended (mentioned in the docs/README) or because of a future change in the code (as a fix to another issue) are not valid issues.

I'm not stating that ERC1155 is less valuable or less popular. But looking at it again, to become a High severity it's very important that material funds are risked of being lost. Which is not the case here.

sashik-eth

> I'm not stating that ERC1155 is less valuable or less popular. But looking at it again, to become a High severity it's very important that material funds are risked of being lost. Which is not the case here.

What do you define as material funds? Only ERC20 tokens and native ETH? In my opinion, the ERC1155 token can also be a material fund or hold access to other tokens or ETH.

Evert0x

@sashik-eth but ERC1155 are not lost in this scenario, that just can not be used in the protocol.

nevillehuang

Hi @sashik-eth, there is no loss of funds here, just a permanent DoS of intended functionality of transferring ERC1155 tokens, so in favor of medium severity.

AhmadDecoded

@nevillehuang my point was, strategies are highly customizable and as allo plan in future they would approved community developed strategies too, so you never know the implementation and the problems it may lead to.

Evert0x

Result: Medium Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- ahmaddecoded: accepted

AhmadDecoded

@Evert0x i think the excluded tag should be removed.